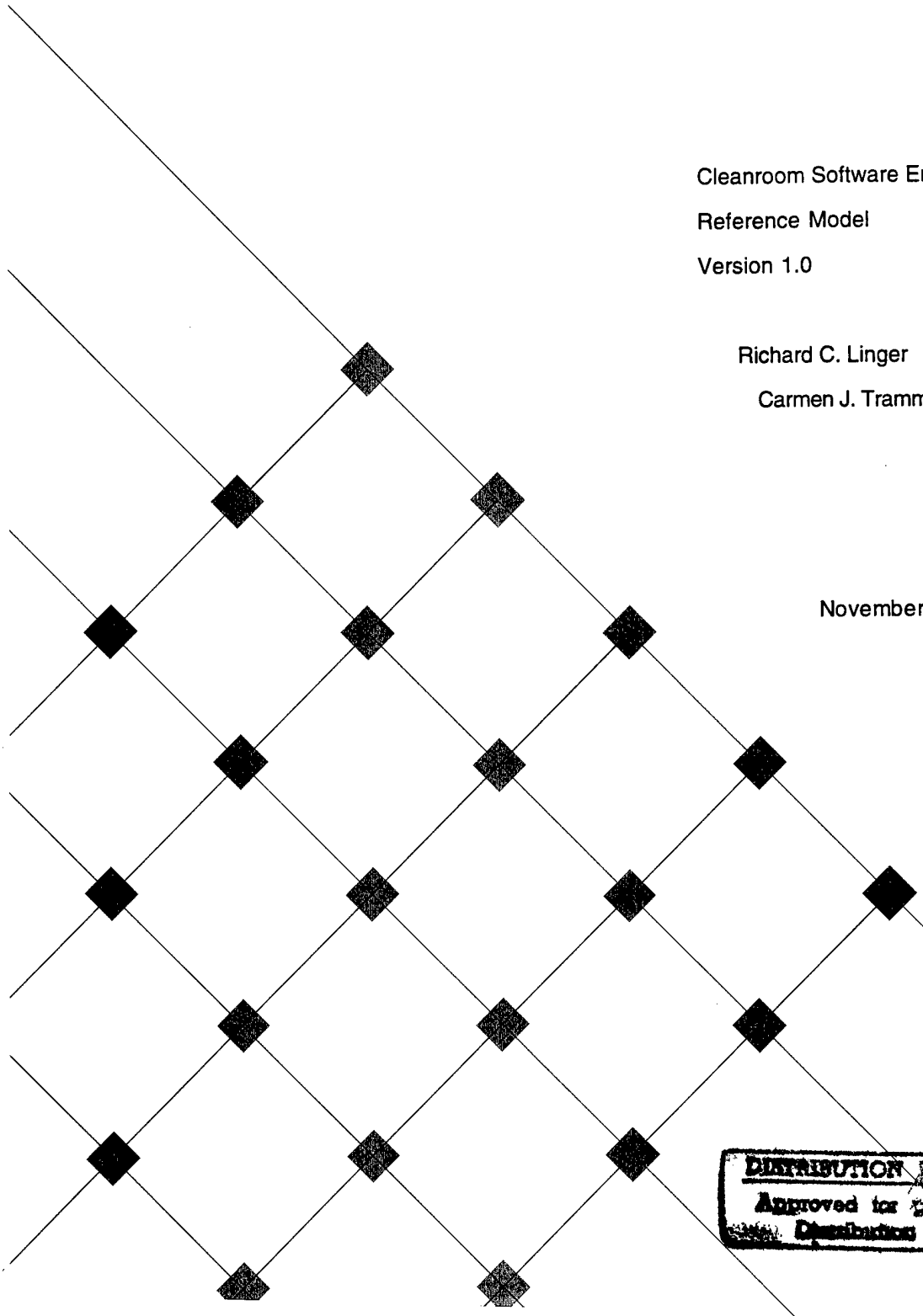
Technical Report
CMU/SEI-96-TR-022
ESC-TR-96-022

Carnegie-Mellon University
Software Engineering Institute

Cleanroom Software Engineering Reference Model Version 1.0

Richard C. Linger
Carmen J. Trammell

November 1996



Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

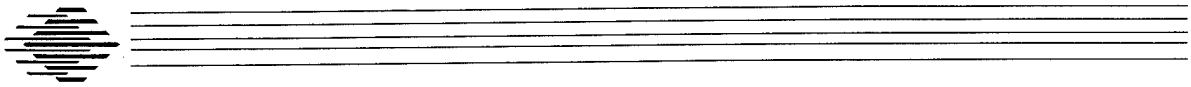
In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of, "Don't ask, don't tell, don't pursue," excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Technical Report
CMU/SEI-96-TR-022
ESC-TR-96-022
November 1996

Cleanroom Software Engineering Reference Model
Version 1.0



Richard C. Linger

Software Engineering Institute

Carmen J. Trammell

University of Tennessee

Process Program

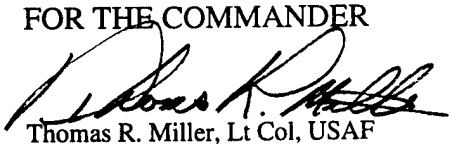
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Contents

Portions of this Technical Report will appear as part of a book in the SEI series in Software Engineering, entitled *Cleanroom Software Engineering*, by Richard C. Linger and Carmen J. Trammell. The book will be published by the Addison-Wesley Publishing Company, Reading, Ma.

List of Figures

Acknowledgments

The authors express appreciation to the many reviewers of this document for their careful review and excellent suggestions. The reviewers included Ingrid Biery, Earl Billingsley, Philip Hausler, Alan Hevner, David Kelly, Ara Kouchakdjian, Don O'Neill, Rose Pajerski, Jesse Poore, Dave Pearson, Ron Radice, Kirk Sayre, Wayne Sherer, Alan Spangler, and Gwen Walton. The authors express special thanks to David Kelly and Wayne Sherer for their extensive review and comments.

1 The Cleanroom Software Engineering Reference Model

Cleanroom software engineering is a theory-based, team-oriented process for development and certification of high-reliability software systems under statistical quality control [Mills 92, Linger 93, Linger 94]. A principal objective of the Cleanroom process is development of software that exhibits zero failures in use. The Cleanroom name is borrowed from hardware Cleanrooms, with their emphasis on rigorous engineering discipline and focus on defect prevention rather than defect removal. Cleanroom combines mathematically-based methods of software specification, design, and correctness verification with statistical, usage-based testing to certify software fitness for use. Cleanroom projects have reported substantial gains in quality and productivity.

This report defines the Cleanroom Software Engineering Reference Model, or CRM. The CRM is expressed in terms of a set of 14 Cleanroom processes and 20 work products. It is intended as a guide for Cleanroom project management and performance, process assessment and improvement, and technology transfer and adoption. The remainder of the report is organized into the following sections:

Section 2: Cleanroom Software Engineering Overview

This section describes characteristics of Cleanroom project performance.

Section 3: Cleanroom Software Engineering Technology

This section introduces the technologies embodied in the Cleanroom processes.

Section 4: Cleanroom Software Engineering Processes

This section defines the 14 Cleanroom processes, organized into categories for project management, software specification, software development, and software certification.

Section 5: Cleanroom Software Engineering Work Products

This section defines the purpose and content of the 20 work products produced by the Cleanroom processes.

The material in this report is not intended to teach the reader how to practice Cleanroom software engineering; that requires attending courses on Cleanroom management and technology. Rather, the intent of this document is to define a reference model that embodies the principal processes and methods of Cleanroom as a guide to project performance by trained teams. It also serves as a baseline for continued evolution of Cleanroom practice.

The scope of the CRM is software management, specification, development, and testing. A number of related activities are not addressed in the reference model, such as marketing, distribution, installation, customer support, and other essential aspects of product success.

As use of the Cleanroom process grows, interest in its relationship to the Capability Maturity Model (CMM[sm]) for Software has increased. The CRM has been mapped to the Key Process Areas (KPAs) of the CMM for Software. The mapping is described in the following SEI Technical Report:

Linger, Richard C.; Paulk, Mark C.; & Trammell, Carmen J. *Cleanroom Software Engineering Implementation of the CMM for Software* (CMU/SEI-96-TR-023). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1996.

Cleanroom provides an effective implementation of much of the CMM for Software. The Cleanroom Reference Model is a framework for defining an organization- or project-specific Cleanroom process, and the CMM mapping of the reference model provides linkage to the CMM Key Process Areas. Their combination represents an "enriched CMM" and an "enriched Cleanroom process" that incorporates substantial management, organizational, and engineering capability.

[sm] CMM and Capability Maturity Model are service marks of Carnegie Mellon University.

2 Cleanroom Software Engineering Overview

2.1 Developing Software Under Statistical Quality Control

The Cleanroom process embeds software development and testing within a statistical quality control framework. Mathematically-based software development processes are employed to create software that is correct by design [Linger 94], and statistical usage testing processes are employed to provide inferences about software reliability [Mills 92]. This systematic process of assessing and controlling software quality during development permits certification of software fitness for use at delivery.

The value of a process under statistical quality control is well illustrated by modern manufacturing processes where the sampling of output is directly fed back into the processes to control quality. Once the discipline of statistical quality control is in place, management has objective visibility into the software development process and can control process changes to control product quality.

Key characteristics of the Cleanroom process are an incremental development life cycle and independent quality assessment through statistical testing. The development life cycle starts with a specification that not only defines function and performance requirements, but also identifies operational usage of the software and a nested sequence of user-function subsets that can be developed and tested as increments which accumulate into the final system. Disciplined software engineering methods provide design and verification techniques required to create correct software. Correctness verification by development teams is used to identify and eliminate defects prior to any execution of the software.

Software execution is controlled by an independent certification team that uses statistical testing methods to evaluate software quality. Statistical testing results in objective quality certification of software at delivery and provides a scientific basis for generalizing reliability estimates to operational environments.

2.2 Cleanroom Application and Results

The Cleanroom processes can be applied to development of new software systems and evolution of legacy systems:

> **New Systems.** The Cleanroom processes provide a rigorous management and technical framework for developing new software systems under intellectual control. Theory-based processes for specification, design, and verification produce software that exhibits very high quality at the inception of testing. Incremental development permits early quality assessment through statistical testing and user feedback on system function, and avoids risks associated with component integration late in the life cycle.
>
> **Legacy Systems.** Modifications and extensions to legacy systems can be developed with the Cleanroom processes. Components of legacy systems can be re-engineered to Cleanroom quality through use of structuring, design abstraction, correctness verification, and statistical testing techniques.

Cleanroom is language-, environment-, and application-independent, and has been used to develop and evolve a variety of systems, including real-time, embedded, host, distributed, workstation, client-server, and microcode systems. Cleanroom supports prototyping and object-oriented development [Ett 96], and enables reuse through precise definition of common services and component functional semantics, and certification of component reliability.

The Cleanroom process has been demonstrated in software development projects in industry, as well as in NASA and the DoD STARS (Software Technology for Adaptable, Reliable Systems) program. Experience has shown substantial improvements over traditional results [Basili 94, Hausler 94, Linger 94]:

> **Quality.** Improvements of 10 to 20X and substantially more over baseline performance have been reported. Failures in field use have been greatly reduced over prior experience. For example, IBM developed an embedded, real-time, bus architecture, multiple-processor device controller product that exhibited no failures in three years use at over 300 customer locations [Linger 94].

Productivity. Significant improvements over baseline performance have been reported. For example, an Ericsson Telecom project to develop a 374 KLOC operating system reported gains of 1.7X in development productivity [Hausler 94], and an IBM project to develop a network management and outage avoidance product reported a 2X improvement in development productivity [Hausler 92]. The US Army Picatinny Arsenal STARS demonstration project reported a 4.6X productivity gain [Sherer 96].

Life cycle costs. Reductions in life cycle costs through decreases in testing, error correction, and maintenance have been reported. For example, IBM developed a COBOL structuring product that exhibited just seven minor errors in the first three years of field use, all simple fixes, with a corresponding drop in maintenance costs compared to baselines for similar products [Linger 88].

Return on investment. Reports have shown that Cleanroom adoption costs can be recovered on the first project through increased effectiveness in development and testing and reduced error correction and rework. For example, a 21 to 1 return-on-investment in Cleanroom technology introduction has been reported by the Picatinny Arsenal STARS Cleanroom project [Sherer 96].

There are other benefits of the Cleanroom process that are less easily quantified. Statistical testing provides project managers with scientific measures of software quality for objective decision making on whether and when to stop testing and release products. These measures also provide projections of software quality in operational use. In combination with incremental development, this fine-grained measurement process substantially improves the predictability of software development. Increases in job satisfaction for Cleanroom teams have also been reported [Sherer 96]. These increases are due in part to improved communication and coordination among team members, based on the shared understandings and uniform work products that Cleanroom methods afford.

3 Cleanroom Software Engineering Technology

The Cleanroom processes employ the following technologies for project management and software specification, design, and testing.

3.1 Incremental Development Life Cycle

Cleanroom management is based on development and certification of a pipeline of user-function software increments that accumulate into the final product [Linger 94]. Incremental development enables early and continual quality assessment and user feedback, and facilitates process improvements as development progresses. The incremental approach avoids risks inherent in component integration late in the development cycle. Incremental development permits systematic management and incorporation of requirements changes over the development cycle.

The technical basis for incremental development in Cleanroom is the mathematical property of referential transparency. In the context of software development, this property requires that a specification and its design decomposition define the same mathematical function, that is, the same mapping from the domain of inputs to the range of correct outputs. When this property holds, a design can be shown to be correct with respect to its specification.

In practice, the requirement for referential transparency places constraints on the functional content and order of design decomposition of a software system. User functions are organized for development into a sequence of verifiable and executable software increments, each providing additional function. The functional content of the increments is defined such that they accumulate into the complete set of functions required for the final system. Architectural requirements and risk avoidance strategies place additional constraints on increment content. For correctness, each increment must satisfy its parent specification through the functions it provides combined with the subspecifications it contains for future increments. For statistical testing and certification, each increment can contain stubs as placeholders for future increments to permit execution in the system environment. Each new increment replaces stubs in the evolving system and satisfies the

subspecifications associated with it. In this way, referential transparency is maintained throughout system development. Referential transparency in incremental development is important for maintaining intellectual control in Cleanroom project management.

A more extensive explanation of incremental development is found in [Trammell 96].

3.2 Precise Specification and Design

A key Cleanroom principle is that programs can be regarded as rules for mathematical functions (or relations). That is, programs carry out transformations from input (domain) to output (range) that can be precisely specified as function mappings. Programs can be designed by decomposing their function specifications, and can be verified by abstracting and comparing their designed functions to their function specifications for equivalence. This concept is scale-free, with application ranging from large specifications for entire systems down to individual control structures (sequence, ifthenelse, whiledo), and to every intermediate decomposition and verification along the way.

Three special types of mathematical functions are important in Cleanroom development because of their correspondence to useful system views, and their interrelationships in a stepwise decomposition and verification process. These forms are known as black box, state box, and clear box, and collectively, as box structures [Mills 86]. Box structures are used extensively by Cleanroom specification and development teams. Other methods may also be used to implement the Cleanroom principles of mathematically-based software specification and development; the box structure method is used in this reference model due to its prominence and proven capabilities in Cleanroom practice.

Box structures map system stimuli (inputs) and the stimulus histories (previous inputs) into responses (outputs). A black box defines the required external behavior of a software system or system part in all possible circumstances of use. The transition function of a black box is

((current stimulus, stimulus history) --> response).

That is, a black box maps the current stimulus into a response that also depends on the history of stimuli received. For example, given a stimulus of 5, a hand

calculator will produce a response of 175 if the stimulus history is C 1 7 (C for Clear), but a response of 5 if the history is C 1 7 +. The stimulus is the same in both cases, but the histories of stimuli are different, leading to different responses. A black box definition is state-free and procedure-free, referencing only external, user-observable stimuli and responses. Black box definitions are often given in tables with columns for current stimulus, conditions on history, and responses. Abstraction techniques—for example, conditions on stimulus histories—permit compact descriptions in scaling up to large systems.

A state box is derived from and verified against a corresponding black box. The state box transition function is

((current stimulus, current state) --> (response, new state)).

That is, a state box maps the current stimulus and the current state into a response and a new state. In the state box, the stimulus history of the black box is replaced by retained state data necessary to achieve black box behavior. A state box definition is procedure-free, and isolates and focuses on state invention. State box definitions are often given in tables with columns for current stimulus, current state, response, and new state.

A clear box is derived from and verified against a corresponding state box. The clear box transition function is

((current stimulus, current state) --> (response, new state)) by procedures.

In the clear box, the procedures required to implement the state box transition function are defined, possibly introducing new black boxes for further decomposition into state and clear boxes. That is, a clear box is a program, or set of programs, that implements the state box and introduces and connects operations in a program structure for decomposition at the next level. Such connections are critical to maintaining intellectual control in large-scale software development.

Box structures can be applied to a variety of decomposition strategies, including functional, object oriented, etc. In the object oriented case, the black box defines the behavior specification of an object, the state box defines its data encapsulation, and the clear box defines its procedural services or methods [Ett 96]. Box structures also provide a systematic framework for incorporating reused and COTS software.

A principal value of box structures lies in the referential transparency between box decompositions that helps maintain intellectual control in large-scale software developments. Box structures play a primary role in function specification, architecture specification, and increment design.

A full explanation of box structure specification and design is given in [Mills 86].

3.3 Correctness Verification

All Cleanroom-developed software is subject to function-theoretic correctness verification by the development team prior to release to the certification test team. The function-theoretic approach permits development teams to completely verify the correctness of software with respect to specifications. A Correctness Theorem defines conditions to be met for achieving correct software [Linger 79]. These conditions are verified in mental/verbal proofs of correctness in development team reviews. Programs contain an infinite number of paths that cannot all be checked by path-based inspections or software testing. However, the Correctness Theorem is based on verifying individual control structures (sequence, ifthenelse, whiledo, etc.) rather than tracing paths. Because programs contain a finite number of control structures, the Correctness Theorem reduces verification to a finite number of checks, and permits all software logic to be verified in possible circumstances of use. The verification step is remarkably effective in eliminating defects, and is a major factor in the quality improvements achieved by Cleanroom teams. The Correctness Conditions defined by the Theorem for fundamental control structures are given in the Correctness Verification Process in this document.

A full explanation of function-theoretic verification is given in [Linger 79].

3.4 Statistical Testing and Software Certification

The set of possible executions of a software system is an infinite population. All testing is really sampling from that infinite population. No testing process, no matter how extensive, can sample more than a minute fraction of all possible executions of a software system. If the sample embodied in a set of test cases is a random sample based on projected usage, valid statistical estimates of software quality and reliability for that usage can be obtained [Mills 92, Whittaker 93].

Statistical usage testing [Poore 95a, Trammell 95, Walton 95a] of a software system produces scientific measures of product and process quality for management decision-making, just as has been done in hardware engineering for decades.

The objective of the certification team is to provide scientific certification of software fitness for use, not to "test in" quality (an impossible task). The certification team creates usage models which define all possible scenarios of use of the software, together with their probabilities of occurrence. Multiple models can be defined to address different usage environments, or to provide independent certification of stress situations or infrequently used functions with high consequences of failure. Usage can be defined in Markov models that permit substantial management analysis and simulation of test operations prior to software development, as well as automatic test case generation [Walton 95a, Whittaker 93]. Other methods may also be used to implement the Cleanroom principles of statistically-based software testing and certification; the Markov approach is used in this reference model due to its prominence and proven capabilities in Cleanroom practice.

Following correctness verification, software increments are delivered to the certification team for first execution. If required, other forms of testing can be applied prior to statistical testing. Test cases are randomly generated from the usage models, so that every test case represents a possible use of the software as defined by the models. Objective statistical measures of software reliability and fitness for use can be computed based on test results for informed management decision-making. Because statistical usage testing tends to detect errors with high failure rates, it is an efficient approach to improving software reliability.

A more extensive discussion of statistical certification testing can be found in [Walton 95a, Whittaker 93, Whittaker 94a].

3.5 Software Reengineering

Non-Cleanroom-developed software can be incorporated into Cleanroom projects. Such software may require reengineering to enable developers to maintain intellectual control and to achieve Cleanroom levels of quality and reliability.

Unstructured software can be transformed into structured form for improved understandability and maintenance through application of the Structure

Theorem [Linger 79]. The constructive proof of the theorem defines a systematic procedure for transforming unstructured logic into function-equivalent structured form. The procedure can be carried out manually, or fully or partially automated if large quantities of software must be structured.

Missing or incomplete designs and specifications of existing structured programs can be recovered and documented through systematic analysis and abstraction based on function-theoretic techniques. An example of function abstraction for recovery of specifications is given in the Software Reengineering Process in this document.

A full explanation of program structuring and function abstraction is given in [Linger 79].

4 Cleanroom Software Engineering Processes

4.1 The Cleanroom Processes

This section defines the 14 processes that comprise the practice of Cleanroom software engineering. These processes form a comprehensive guide to Cleanroom project performance for software teams trained in Cleanroom methods. They embody the Cleanroom technologies described in the previous section. The processes are listed below in four sets corresponding to the principal functions in Cleanroom projects. Work products produced in each of the processes are shown in italics. The work products are discussed in the process definitions and summarized in Section 5.

Cleanroom Management Processes

Project Planning Process
Cleanroom Engineering Guide
Software Development Plan
Project Management Process
Project Record
Performance Improvement Process
Performance Improvement Plan
Engineering Change Process
Engineering Change Log

Cleanroom Specification Processes

Requirements Analysis Process
Software Requirements
Function Specification Process
Function Specification
Usage Specification Process
Usage Specification
Architecture Specification Process
Software Architecture
Increment Planning Process
Increment Construction Plan

Cleanroom Development Processes

Software Reengineering Process

Reengineering Plan

Reengineered Software

Increment Design Process

Increment Design

Correctness Verification Process

Increment Verification Report

Cleanroom Certification Processes

Usage Modeling and Test Planning Process

Usage Models

Increment Test Plan

Statistical Test Cases

Statistical Testing and Certification Process

Executable System

Statistical Testing Report

Increment Certification Report

4.2 Cleanroom Process Flow

Figure 1 depicts a typical flow and interaction among the Cleanroom processes. The four management processes affect all Cleanroom operations, and are shown across the top of the figure at (1-4). In the Project Planning Process, which may be repeatedly invoked during the project, the team tailors the Cleanroom processes for the project environment and creates and maintains software development plans. These plans are used in the Project Management Process for managing and controlling incremental development and certification. The Performance Improvement Process is used to continually assess project performance in applying the Cleanroom processes, and to identify and implement improvements. The Engineering Change Process provides necessary configuration management and engineering discipline for all change activity.

At (5), the Architecture Specification Process likewise spans the entire life cycle, in analyzing architectural assets and defining architectural structures and strategies for software development at multiple levels of abstraction.

At (6), the Requirements Analysis Process is used to create an initial definition of customer requirements. This definition is then expressed in precise terms in the Function Specification Process (7), producing a specification of required external behavior, and the Usage Specification Process (8), producing a specification of users, usage environments, and patterns of use of the software system. The Increment Planning Process (9) allocates specified software functions into a set of increments, and schedules their development and certification within the structure of the overall project schedule.

Prior to development and certification of each increment, the analysis and specification processes at (6-8) may be revisited to validate requirements and incorporate feedback from customer assessment of prior increments. The processes at (10-14) are invoked iteratively for development and certification of successive increments. At (10-12), the Software Reengineering Process prepares existing software for use in an increment, and the Increment Design and Correctness Verification Processes are employed to develop the detailed design and code for an increment, and to verify their correctness. At (13), the Usage Modeling and Test Planning Process is used to create required usage models and test plans. Finally, at (14), the Statistical Testing and Certification Process is employed to test an increment using test cases generated from the usage models, and to assess an increment's fitness for use in terms of statistical measures. Certified increments are provided to the customer for assessment and requirements validation.

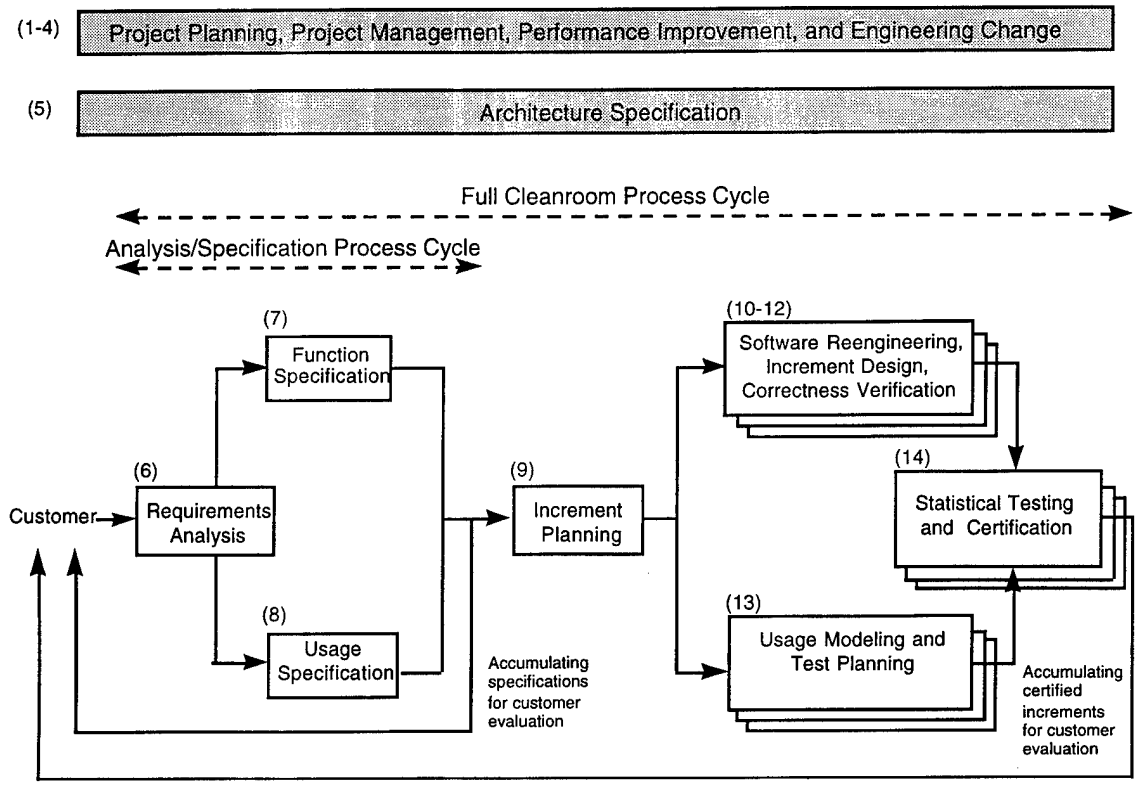


Figure 1. Cleanroom Process Flow

4.3 Cleanroom Process Application: Scenarios of Use

Cleanroom is a well-defined yet flexible engineering process. It scales up for development of systems of arbitrary size and complexity, and scales down for development of small, stand-alone applications. The Cleanroom processes and the technology they embody are intended to be applied by management in flexible ways based on the project environment and objectives, as the following examples illustrate.

1. Unprecedented Systems

When customer requirements are uncertain, the process flow of Figure 1 may undergo several rapid traversals to develop and evolve prototypes for customer assessment and requirements elicitation. In this case, certain process steps may be scaled back or deferred if limited-scope, short-lived prototypes are intended, or, alternatively, process steps may be fully implemented if broader-scope prototypes are intended to be scaled up to full systems.

2. Embedded Software

Embedded software systems that perform critical functions often face very high failure costs in terms of potential product recalls. It can make economic sense in developing such systems to take a more rigorous, granular approach to the Function Specification, Usage Specification, Increment Design, Correctness Verification, Usage Modeling and Test Planning, and Statistical Testing and Certification Processes to help ensure failure-free performance in use.

3. Legacy Systems

Systems in a mature product line may require extensive application of the Software Reengineering, Correctness Verification, Usage Modeling and Test Planning, and Statistical Testing and Certification Processes to ensure that the reused software achieves Cleanroom quality objectives.

4. Large, Complex Systems

Large-scale system developments often involve complex collections of hardware, software, and human components. Such projects typically require extensive systems engineering and analysis for project planning, requirements allocation, system specification, and architecture definition prior to any software development. The set of Cleanroom processes labeled "Analysis/Specification Process Cycle" in Figure 1 can be repeatedly applied to establish and evolve the system structure and project plans for subsequent software development. In addition, large, complex systems usually require a final system certification step.

4.4 Cleanroom Teams

Cleanroom teams have management, specification, development, and certification roles. Teams are typically composed of five to eight people. On small projects, individuals may serve on several teams. On large projects, teams of teams may be required, with hundreds of people involved. In this case, the structure and evolution of the teams can be driven by the structure and evolution of the system under development. Teams can also be organized as integrated product teams, provided the Cleanroom roles for specification, development, and certification are maintained.

Whatever the team organization, the software project manager has overall project responsibility. A chief specification engineer, chief design engineer, and chief certification engineer lead the specification, development, and certification teams, respectively.

The overarching objectives for a Cleanroom project are as follows.

- software project manager: management of Cleanroom processes to achieve software product completion on schedule and within budget
- specification team: specification of required software function and usage in all circumstances of use
- development team: development of fault-free software that implements specified function
- certification team: certification of software fitness for use for specified function and usage

Cleanroom teams interact with a variety of peer organizations depending on the organizational and project context. System engineering and system test organizations may be involved in embedded software projects; standards, procurement, and quality assurance organizations may be involved in large projects; configuration management, documentation, and organizational software engineering process groups will likely be involved in any software project; and so on.

The customer is part of the Cleanroom team as well. The term "customer" may mean external institutional sponsor, internal organizational sponsor, end user, or any other party that is appropriate for defining requirements and evaluating the evolving system.

4.5 Cleanroom Process Definitions

As noted, the Cleanroom Reference Model is a high-level Cleanroom process template that must be tailored for use by a specific organization or project. Each process and work product in the Cleanroom Reference Model is intended to be elaborated through implementation procedures that address specific organizational or project environments and unique development requirements. These implementation procedures are to be documented in the *Cleanroom Engineering Guide.*

The 14 Cleanroom processes are defined below in the following augmented ETVX (Entry, Task, Verification, Exit) format:

Objectives

The Objectives section defines the outcomes of effective process performance.

Participants

The Participants section defines the roles of the participants in the process. The participants may include the performers of tasks, the reviewers of work products, or others who receive information about the status or outcomes of the process.

Entry

The Entry section defines the entry criteria that must be satisfied for the process to be initiated, and lists the work products that must be available as inputs to the process.

Tasks

The Tasks section defines work to be carried out in performing the process. The order of the tasks is generally, but not strictly, sequential. Some tasks may be concurrent with other tasks.

Verification

The Verification section defines steps for verifying that the process has been properly executed, and that the associated work products meet project objectives.

Measurement

The Measurement section defines Cleanroom measures for assessing (1) the performance of the process and (2) the characteristics of the work products. The measures given in the Measurement section are either characteristic of or integral to Cleanroom software engineering. Many other measures not given in the Measurement section may also be useful, or even required in a given project.

Exit

The Exit section defines the exit criteria that must be satisfied for the process to be terminated. The exit criteria generally involve completion and verification of

work products, but may also be given in terms of quantitative or qualitative conditions of work products.

Other Syntactical Elements

Boxed text appears in the process definitions to (1) explain Cleanroom terms and concepts, (2) recommend specific implementation techniques, (3) provide examples, and (4) point to further information. Accordingly, the boxes are labeled Explanation, Recommendation, Example, or Reference.

Work product names are given in italics. The purpose and content of the work products are described in Section 5.

Common Cleanroom Process Elements

The Cleanroom processes have a number of elements in common. These common elements are defined in a single section named Common Cleanroom Process Elements, to avoid repetition and achieve more compact definitions of the Cleanroom processes. The Common Cleanroom Process Elements apply to all of the Cleanroom processes.

Common Cleanroom Process Elements

The common objectives, participants, entry criteria, tasks, verification, measures, and exit criteria in Cleanroom processes are given here as common Cleanroom process elements. That is, these elements should be treated as part of every Cleanroom process. Rather than being restated in each Cleanroom process, the common elements have been "factored out" and stated once.

The people who are responsible for each of the Cleanroom management, specification, development, and certification processes (i.e., the "process owners") should regard the common elements to be included in their process responsibilities.

Common Objectives

Objective 1	Work products created or updated in the process are traceable to the Entry work products from which they were derived.
Objective 2	Defects in work products created or updated in the process are identified through peer review and are eliminated.

Common Participants

In addition to project staff, participants include management, peer organization representatives, and customer representatives as appropriate for review, information, or agreement.

Common Entry

Entry 1	The *Cleanroom Engineering Guide* and the *Software Development Plan* (developed in the Project Planning Process), and the *Project Record* are available. When the process is reentered for changes to work products, the reentry is consistent with the Engineering Change Process and the *Configuration Management Plan.*

Common Tasks

Task 1	Ensure that all participants understand process requirements as documented in the *Cleanroom Engineering Guide.*
Task 2	Create work products in the formats defined in the *Cleanroom Engineering Guide.*
Task 3	Make changes to work products in compliance with the Engineering Change Process and the *Configuration Management Plan.*

Task 4 **Document project activity in the *Project Record.***

Document information that will not be recorded in other work products in the *Project Record*. Specifically, document process beginning and ending dates, staff assignments, process review dates and data, measurements, and other key events and decisions.

Common Verification

Verification 1 **Review the status of the process with management, the project team, peer groups, and the customer.**

These verification activities include confirmation that the process was performed as defined in the *Cleanroom Engineering Guide*.

Verification 2 **Review work products created or updated in the process with the project team.**

Work products are verified against properties defined for them in the *Cleanroom Engineering Guide*. Work products under review are verified to be fully traceable to the work products from which they were derived.

> EXPLANATION: Peer review
>
> Peer review is a key to intellectual control of work by Cleanroom teams. The work of an individual team member is regarded as a draft until there is team consensus that the work is correct and of acceptable quality.
>
> Every Cleanroom work product is peer reviewed, yielding substantial benefits. Differing interpretations of requirements are uncovered, conventions are established, errors are detected, opportunities for economy are identified, understandability is tested, and expertise is shared. The results benefit the project, the product, and the team members alike.

REFERENCE: CMM Peer Reviews and Defect Prevention KPAs

If compliance with these KPAs is an organizational objective, their specific requirements should be reviewed when this verification step is tailored for organizational or project use.

Common Measurement

Measurement 1 **Measure the process.**

Measure process performance in terms such as deviations in resource and schedule actuals from plans.

Measure the effectiveness of a review in terms of the percentage of all defects that are found in the review. These percentages are determined, of course, after execution testing.

EXPLANATION: Effectiveness of reviews

The earlier a product defect is discovered, the less costly it is to fix. If most defects are found early in the development cycle, a great deal of costly rework is avoided. Similarly, if most defects are found late in the development cycle, costly rework is incurred.

The distribution of total defects across all reviews is an indication of the relative effectiveness of reviews. A more precise measure is the distribution of defects across all reviews in which they could have been found, i.e., the distribution of requirements defects identified in requirements and succeeding reviews; the distribution of specification defects identified in specification and succeeding reviews, etc.

Measurement 2 **Measure the product.**

Measure size and stability of work products that define the software (i.e., the *Software Requirements*, the *Function*

Specification, *Usage Specification*, and *Software Architecture*, the *Usage Models*, the *Increment Design*, and the *Executable System*).

Measure the quality of work products that define the software in terms of the percentage of execution failures that are traced to defects in the work products. These percentages are determined, of course, after execution testing.

EXPLANATION: Quality of work products

If an execution failure is traced to a defect in a work product, the quality of that work product is suspect. The causes of all execution failures should be traced to the work products in which they originate. The resulting distribution of defects reflects the relative quality of the work products.

Common Exit

Tasks and Verification activities have been completed, and the *Project Record* has been updated.

Project Planning Process

A Cleanroom management process

The purpose of the Project Planning Process is to 1) tailor the Cleanroom Reference Model (or the organizational reference model) for the project, 2) define and document plans for the Cleanroom project, and 3) review the plans with the customer, the project team, and peer groups for agreement.

The work products of the Project Planning Process are the *Cleanroom Engineering Guide* and the *Software Development Plan.* Both documents are revised as necessary during the project to accommodate customer needs and project performance.

The *Cleanroom Engineering Guide* defines a tailoring of the Cleanroom processes to meet project-specific process requirements.

The *Software Development Plan* is the repository for project management plans, including mission, organization, work products, schedules, resources, measurements, reuse analysis, risk analysis, standards, training, and configuration management. The *Software Development Plan* is used in the Project Management Process for task initiation, performance tracking, and quantitative process management.

The *Cleanroom Engineering Guide* and the *Software Development Plan* form the basis for defined, repeatable, managed, and optimizing performance of Cleanroom activities.

Objectives

Objective 1	**Cleanroom software engineering processes are tailored for the project and documented.**
Objective 2	**The software project plans are defined and documented.**
Objective 3	**The customer, the project team, and peer groups agree to the Cleanroom processes and project plans.**

Participants

Project software manager, chief specification engineer, chief development engineer, chief certification engineer, peer organizations, and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1	A new or revised *Statement of Work* and/or *Software Requirements* exists for the software project.
Entry 2	The *Software Development Plan* and/or *Cleanroom Engineering Guide* require revision or elaboration at the beginning of a new increment or as necessary.

Entry work products are available.

Tasks

Task 1

Create the *Cleanroom Engineering Guide.*

Use the *Cleanroom Software Engineering Reference Model* or the organizational Cleanroom Software Engineering

Reference Model, if any, as the basis for defining or revising the project's *Cleanroom Engineering Guide,* including:

1. Project-specific tailoring and refinement of the Cleanroom processes. Define and document clear process implementation guidance for the Cleanroom project.

2. Identification and documentation of facilities, hardware, and software environments and tools to support Cleanroom processes, with guidelines for their use.

> REFERENCE: CMM Organizational Process Definition, Integrated Software Management, Software Product Engineering, and Software Quality Management KPAs
>
> If compliance with these KPAs is an organizational objective, their specific requirements should be reviewed when the Cleanroom processes are tailored for organizational or project use.

Task 2 **Create the *Software Development Plan.***

> REFERENCE: CMM Software Project Planning KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the *Software Development Plan* is developed.

Use the *Statement of Work* and/or *Software Requirements* to define or revise the *Software Development Plan,* including the following plans:

1. *Project Mission Plan*: Define the overall mission, goals, and objectives of the software product and the Cleanroom development project.

2. *Project Organization Plan:* Define the structure, responsibilities, and relationships of the Cleanroom project organization. Identify points of contact in customer and peer organizations.

REFERENCE: CMM Intergroup Coordination KPA

If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the *Project Organization Plan* is developed.

3. *Work Product Plan:* Define the Cleanroom work products and customer deliverables to be produced during the project.

4. *Schedule and Resource Plan*: Define estimates for overall schedules, milestones, and budgets. Define staffing, system, and other resource requirements. These estimates will be refined in the Increment Planning Process.

5. *Measurement Plan*: Define product and process measures for managing the project, including goals for Cleanroom software certification and standards for statistical process control. Define the use of measures in project reviews and decision making.

EXPLANATION: Quantitative management decisions

A quantitative basis for management decisions regarding product quality and process control is a hallmark of Cleanroom. The organizational data base of project measures that accumulates over time becomes increasingly useful in planning and management. A historical baseline of product measures (e.g., size, stability, and quality) and process measures (e.g., conformance to plans and effectiveness of reviews) provides a basis for 1) estimating schedules, budgets, and resources, 2) defining process control standards for work in progress, and 3) defining certification goals for increment and product certification.

REFERENCE: CMM Quantitative Process Management KPA

If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the *Measurement Plan* is developed.

6. *Reuse Analysis Plan*: Define methods for identifying and evaluating opportunities to reuse existing assets and create new reusable assets. Reusable assets include domain models, reference architectures, software specifications, designs, implementations, and usage models. Define specific opportunities for reuse.

7. *Risk Analysis Plan*: Define methods for identifying and managing risks throughout the project. Define specific management and technical risks associated with the project.

8. *Standards Plan*: Identify and define the application of external standards that will be used in the project.

9. *Training Plan*: Identify project training requirements, including training in the application domain, development environments, and Cleanroom technology and processes.

> REFERENCE: CMM Training Program KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the *Training Plan* is developed.

10. *Configuration Management Plan*: Identify the work products to be maintained under configuration control. Define procedures for change management and configuration control of the work products.

> REFERENCE: CMM Software Configuration Management KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the *Configuration Management Plan* is developed.

Verification

Verification 1 **Review the *Cleanroom Engineering Guide* for agreement.**

Review the *Cleanroom Engineering Guide* with the project team and peer groups to obtain commitments to Cleanroom processes and team performance objectives.

Review the *Cleanroom Engineering Guide* with the customer; modify and re-review as necessary to obtain concurrence.

Verification 2 **Review the *Software Development Plan* for agreement.**

Review the *Software Development Plan* with the project team and peer groups to obtain commitments to project plans and schedules.

Review the *Software Development Plan* with the customer; modify and re-review as necessary to obtain concurrence.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria are satisfied.

The *Software Development Plan* and the *Cleanroom Engineering Guide* have been completed and reviewed with the project team, peer organizations, and the customer, and commitments have been obtained.

Project Management Process

A Cleanroom management process

The purpose of the Project Management Process is to manage the Cleanroom project to deliver the software on schedule and within budget. Management responsibilities include managing interaction with the customer and peer organizations; establishing and training Cleanroom teams; initiating, tracking, and controlling planned Cleanroom processes; eliminating or reducing risks; revising plans as necessary to accommodate changes and actual project results; and continually improving Cleanroom team performance.

Cleanroom management is guided by quantitative measurements of process and product performance as defined in the *Measurement Plan*—in particular, the measurements produced by statistical testing and certification of successive increments throughout the project life cycle.

Overall project processes, schedules, and resource allocations are managed according to the *Schedule and Resource Plan.* The *Increment Construction Plan,* created in the Increment Planning Process, provides detailed schedules for managing increment development and certification within the overall schedules. The *Risk Analysis Plan* defines risks to be managed.

An important aspect of Cleanroom project management is establishing and enforcing standards of performance for Cleanroom operations. The Cleanroom development process is designed for defect prevention through mathematically-based specification, design, and correctness verification. Development teams are expected to produce fault-free software that implements specified behavior. The Cleanroom testing process is designed for scientific certification of software fitness for use through statistical testing. Certification teams are expected to produce valid statistical estimates of software quality, not attempt to test in quality.

Objectives

Objective 1 **The project plan is implemented using a tailored Cleanroom process, and schedules, budgets, and quality objectives are met.**

Objective 2 **The project is performed under statistical quality control.**

Objective 3 **The delivered software meets the customer's requirements and is statistically certified to be fit for its intended use.**

Participants

Project software manager, specification team, development team, certification team, peer organizations, and the customer.

Entry

The process begins when the entry criteria are satisfied.

The *Software Development Plan* and the *Cleanroom Engineering Guide* have been completed, reviewed, and agreed to by the project team, peer groups, and the customer.

All project work products are available for use in this process as they are developed.

Tasks

Task 1 **Manage customer interaction.**

Establish and maintain communication with points of contact in customer organizations. Maintain all information received from the customer.

Conduct reviews with the customer on project status and plans.

Establish procedures for customer evaluation of completed software increments.

Task 2 **Manage peer organization interaction.**

Establish and maintain communication with points of contact in peer organizations.

Conduct reviews with peer organizations on project status and plans.

Task 3 **Form, staff, and train the Cleanroom teams.**

Create a Cleanroom organizational structure composed of four functions:

1. Management team led by the project software manager
2. Specification team led by the chief specification engineer
3. Development team led by the chief development engineer
4. Certification team led by the chief certification engineer

Provide team training in the application domain, development environment, and Cleanroom software engineering as defined in the *Training Plan.*

Task 4 **Initiate Cleanroom processes.**

Initiate Cleanroom processes defined in the *Cleanroom Engineering Guide,* as required by *Software Development Plan*—in particular, the processes, schedules, and resource allocations defined in the *Schedule and Resource Plan* and the *Increment Construction Plan.* Document process initiation in the *Project Record.*

Task 5 **Monitor Cleanroom process performance and work products through measurement and take corrective action as necessary.**

Record measurements of process and product performance over the life of the project as defined in the *Measurement Plan.*

Use measurements to monitor performance with respect to plans. Inspect work products to assess adherence to the process. Measurements from the Correctness Verification and Statistical Testing and Certification Processes are especially important in assessing product quality and team performance.

Address performance shortfalls or windfalls. Identify schedule and quality deviations, and implement corrective actions. Revise project plans as necessary through the Project Planning, Increment Planning, Project Management, and Performance Improvement Processes.

Maintain consistency among related work products produced by the Cleanroom processes in accordance with the *Configuration Management Plan.*

> REFERENCE: CMM Software Project Tracking and Oversight and Quantitative Process Management KPAs
>
> If compliance with these KPAs is an organizational objective, their specific requirements should be reviewed when this task is tailored for organizational or project use.

Task 6 **Manage project risks.**

Identify and manage risks according to the *Risk Analysis Plan.* Use the Cleanroom incremental development and certification process as a risk management strategy.

Task 7 **Manage Cleanroom team performance.**

Manage team performance and implement improvements in Cleanroom processes defined in the *Performance Improvement Plan.*

Verification

See Common Cleanroom Process Elements on page 21.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria are satisfied.

The Cleanroom software development project has been completed, and the *Project Record* has been completed.

Performance Improvement Process

A Cleanroom management process

The purpose of the Performance Improvement Process is to 1) continually evaluate and improve team performance in the application of Cleanroom and other software technologies and processes and 2) to evaluate and introduce appropriate new technologies and processes.

Frequent and objective evaluation of team performance is essential to achieve continuous improvement. Causal analysis of deviations from plans can provide early identification of risks. Causal analysis of faults found through the Correctness Verification and the Statistical Testing and Certification Processes can identify areas that require improvement through better process definition, increased emphasis, and/or additional training.

Process and product evaluations in review, verification, testing, and certification activities provide an objective basis for justifying and targeting process improvements. Improvements can be introduced within a project at specific milestones, such as initiation of successive increments, and across projects through coordinated organizational process improvement.

New technologies and processes can be evaluated in pilot applications for their impact on productivity and quality, and introduced in a systematic manner if proven effective.

Objectives

Objective 1 — **The performance of the Cleanroom team is continuously improved.**

Objective 2 — **New Cleanroom and other software technologies and processes are evaluated and introduced as appropriate, and produce improvement in process performance and product quality.**

Participants

Project software manager, specification team, development team, certification team.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — A process step, a software increment, or a work product has been completed and a team review is scheduled.

Entry 2 — New Cleanroom technologies and/or processes are to be evaluated.

Entry 3 — Shortfalls in Cleanroom process performance or work product quality have been identified.

Supporting work products are available.

The *Increment Verification Report, Statistical Testing Report, Increment Certification Report,* and *Engineering Change Log,* if any, define measures of Cleanroom process performance and software product quality.

New Cleanroom or other software technology and process documentation, if any, may be evaluated.

Tasks

Task 1

Evaluate Cleanroom team performance and develop improvement plans.

Evaluate project performance with respect to the *Software Development Plan* and apply trend and causal analysis to deviations.

Apply causal analysis to faults found in the Correctness Verification and Statistical Testing and Certification Processes to identify process steps in which they were introduced and to determine why they occurred.

Compare process and product measurements with historical team performance to assess process control.

Develop plans to improve team performance, including additional training, improved tools and procedures, and revised Cleanroom processes, and document the plans in the *Performance Improvement Plan.*

> REFERENCE: CMM Process Change Management KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when this task is tailored for organizational or project use.

Task 2

Evaluate new technologies and processes and develop implementation plans.

Identify new Cleanroom and other software technologies and processes, and evaluate their impact on current Cleanroom processes. Conduct experiments in the project environment to measure their effectiveness.

Develop plans for introduction of proven new technologies and processes, and document them in the *Performance Improvement Plan.*

Schedule new technology and process introductions for the start of subsequent increments or subsequent projects, as appropriate.

> REFERENCE: CMM Technology Change Management KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when this task is tailored for organizational or project use.

Verification

See Common Cleanroom Process Elements on page 21.

Measurement

Measurement 1 **Measure performance improvement.**

Assess the effect of process and technology changes by examining trends in measures defined in the *Measurement Plan* across successive increments.

Exit

The process is complete when the exit criteria are satisfied.

The *Performance Improvement Plan* has been applied and the recommendations have been implemented. Any changes, such as revisions to the *Software Development Plan* or *Cleanroom Engineering Guide,* have been completed.

Engineering Change Process

A Cleanroom management process

The purpose of the Engineering Change Process is to plan and perform additions, changes, and corrections to work products in a manner that preserves correctness and is consistent with the *Configuration Management Plan*.

Proposed changes to work products are documented in the *Engineering Change Log*. The status of the changes (e.g., proposed, approved, rejected, scheduled, in progress, completed) is updated throughout the process.

Changes are made with full engineering rigor and discipline using the Cleanroom processes. The highest level of specification or design affected by a change is identified as the starting point for any respecification, redesign, reverification, recertification, and any other revision activity.

Objectives

Objective 1 — **Additions and changes to work products occur in a manner that preserves correctness and is consistent with the *Configuration Management Plan*.**

Participants

Software project manager, and specification team and/or development team and/or certification team.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — An *Increment Verification Report, Statistical Testing Report,* or report from field use identifies software faults or failures that require correction.

Entry 2 — New requirements or insights require engineering changes to be made to work products.

Entry work products and these work products are available.

The *Software Development Plan, Increment Construction Plan,* and *Reengineering Plan* may be affected by engineering change activity.

Tasks

Task 1 — **Document proposed engineering changes in the *Engineering Change Log*.**

Task 2 — **Evaluate the impact of proposed engineering changes.**

Analyze the scope and impact of proposed changes on project work products, and approve or reject them based on the analysis.

Task 3 **Identify the Cleanroom processes required to perform the engineering changes.**

Define the Cleanroom process sequencing and scheduling required to perform approved engineering changes, and if necessary, revise the *Software Development Plan, Increment Construction Plan,* and/or *Reengineering Plan.*

Task 4 **Apply the Cleanroom processes to perform the engineering changes.**

Apply Cleanroom processes to incorporate the engineering changes at the highest level of specification affected, reengineer subsequent levels of decomposition, and reverify all affected work products for correctness. Maintain the correctness and integrity of all affected work products as the engineering changes are made, and satisfy the requirements of the ***Configuration Management Plan.***

Verification

Verification 1 **Confirm the consistency of engineering change decisions with the *Configuration Management Plan.***

Measurement

Measurement 1 **Use measurements from other Cleanroom processes.**

Use measurements defined for each Cleanroom process initiated through the Engineering Change Process.

Exit

The process is complete when the exit criteria is satisfied.

The required engineering changes have been completed, the necessary work products have been revised, and the *Engineering Change Log* has been updated.

Requirements Analysis Process

A Cleanroom specification process

The purpose of the Requirements Analysis Process is to 1) define requirements for the software product, including function, usage, environment, and performance, and 2) to obtain agreement with the customer on the requirements as the basis for function and usage specification. The specification team creates the *Software Requirements* document as the repository of all requirements information. Elicitation and analysis of requirements is carried out in close cooperation with the customer and peer engineering organizations, and the requirements are typically documented in user terms.

Requirements analysis may identify opportunities to simplify the customer's initial product concept, and to reveal requirements that the customer has not addressed. Early simplification and clarification of requirements can result in schedule and resource savings throughout the development process.

The *Software Requirements* are the customer's requirements; they are the basis for customer acceptance of the product. The *Software Requirements* are the principal input to the Function Specification and Usage Specification Processes, where they are elaborated into the mathematically complete and consistent form essential to intellectual control over development and certification. These processes in turn produce the *Function Specification* and *Usage Specification*, which serve as the developer's technical specifications for the software product.

Requirements are reconfirmed or clarified throughout the incremental development and certification process. The customer executes completed increments and provides feedback on the evolving system.

Objectives

Objective 1 — **Software requirements, including function, usage, environment, and performance are clearly stated, internally consistent, technically feasible, and testable.**

Objective 2 — **The customer agrees with the software requirements as the basis for software specification.**

Objective 3 — **The software requirements are reconfirmed or clarified at the completion of software increments through customer evaluation.**

> REFERENCE: CMM Requirements Management KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the Requirements Analysis Process is tailored for organizational or project use.

Participants

Specification team and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — The *Statement of Work* or other initial artifact, such as a statement of allocated system requirements, is available.

Entry 2 — Changes, including additions and corrections, to the *Software Requirements* are proposed.

Entry 3 — A completed increment is ready for customer execution and evaluation.

Entry work products and these supporting work products are available.

The *Engineering Change Log* and the *Increment Evaluation Report*, if any, contain customer feedback from increment execution and may identify proposed changes to requirements.

Tasks

Task 1

Define the software requirements.

Understand and analyze the *Statement of Work*, the customer's environment, and the context and mission of the product to be developed.

Define requirements, including software function and usage, hardware and software configurations and environments, interfaces, operational constraints, dependencies, and goals for reliability, capacity, and performance.

EXPLANATION: Sources of requirements

Requirements come from many different sources depending on the nature of the product.

- Software that is part of an embedded system or larger software system will be defined on the basis of allocated requirements from the system of which it is a part.
- A product that is part of a product line may inherit requirements related to architecture, interfaces, standard components, etc.
- Marketing, manufacturing, distribution, and other peer organizations may be a source of requirements.
- Industry standards, regulatory standards, export standards, and other commercial or contractual standards can influence requirements.

All relevant sources of requirements should be identified for the system under development.

EXPLANATION: Prototyping

If the requirements definition is insufficient for software specification, initial software increments can be specified and developed as prototypes to obtain user feedback for establishing the requirements.

Simplify requirements and investigate alternatives to improve usability and reduce development and certification effort.

Document requirements and associated assumptions in the *Software Requirements*.

Task 2

Upon completion of each increment, reconfirm or clarify requirements through customer evaluation of the executable system.

Monitor customer execution and evaluation of completed software increments to confirm existing *Software Requirements* or identify proposed changes.

Verification

Verification 1

Review the evolving *Software Requirements* work product.

Conduct frequent specification team reviews of the evolving *Software Requirements* for clarity, consistency, feasibility, and testability. Make simplification of requirements an explicit objective.

Verification 2

Validate the *Software Requirements* work product with the customer and peer organizations.

Review the *Software Requirements* with the customer and affected peer organizations for agreement on the basis for software specification.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria are satisfied.

The new or changed *Software Requirements* are complete and verified, and approved by the customer as the basis for further development.

EXPLANATION: Formal baselining of requirements

It is often the case that requirements cannot be "baselined" and established as the basis for acceptance of the product by the customer until well into the Function Specification, Usage Specification, and Architecture Specification processes. The Requirements Analysis Process and the aforementioned processes are often concurrent—not sequential—processes.

Function Specification Process

A Cleanroom specification process

The purpose of the Function Specification Process is to 1) specify the complete functional behavior of the software in all possible circumstances of use and 2) obtain agreement with the customer on the specified function as the basis for software development and certification.

The specification team creates the *Function Specification* document to satisfy the software requirements. It expresses the requirements in a mathematically precise, complete, and consistent form. The required behavior of the software for every user scenario, however likely or unlikely to occur, is defined in the specification. The specification is an unambiguous definition of the external behavior of the software. No invention of external behavior should be required in subsequent software development.

The *Function Specification* is based on the *Software Requirements*. Once the specification has been completed and validated, it becomes the definitive statement of functional behavior for the software. The specification defines the capabilities to be created through incremental software development. It also serves as the basis for usage specification and usage model development in incremental software certification.

For large systems, a strategy of incremental specification is usually necessary. In this approach, software increments are iteratively specified, developed, and certified. This permits user feedback on observed increment behavior in execution, and can help to elicit requirements that may have proven difficult to define. The Function Specification Process is ongoing. Whenever the evolving *Function Specification* is sufficient to support increment planning and development of an increment, that development can be initiated.

Objectives

Objective 1 **The required behavior of the software in all possible circumstances of use is defined and documented.**

Objective 2 **The function specification is complete, consistent, correct, and traceable to the software requirements.**

Objective 3 **The customer agrees with the function specification as the basis for software development and certification.**

Participants

Specification team and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 The *Software Requirements* have been partially or fully completed.

> EXPLANATION: Incremental function specification
>
> All software requirements must eventually be defined to permit complete function specification. Often, all requirements are not fully understood at the outset, and a strategy of incremental function specification based on partial requirements definition may be necessary.
>
> In large-scale developments, incremental function specification is often a desirable strategy for pacing development, maintaining intellectual control, and eliciting customer feedback.

Entry 2 The *Function Specification* requires revision for changes to the *Software Requirements,* or for changes from increment specification, development, or certification.

Entry work products and these supporting work products are available.

The *Engineering Change Log* describes proposed changes. The *Usage Specification*, if any, is used as a check on the completeness and consistency of the *Function Specification*.

Tasks

Task 1 **Define the format and notation of the *Function Specification*.**

> EXAMPLE: *Function Specification* format
>
> The mathematical definition of a black box specification prescribes certain elements whose format must be specified. For example, a black box specification can be formatted as tables (with columns for current stimulus, conditions on stimulus history, and responses), enumerations of input sequences and responses, disjoint conditional rules, or other formalisms appropriate to the application. The notation definition should include project conventions for naming and typing.

Task 2 **Define all software boundaries and stimulus/response interfaces with hardware, other software, and human users.**

Specify stimuli from hardware devices and associated responses and protocols.

Specify stimuli from external software and associated responses, including formats of files and messages.

Specify stimuli from user interfaces and associated responses, including details of presentation and interaction.

> RECOMMENDATION: Specification of human user interfaces
>
> The details of human user interfaces should be established during function specification, not deferred for completion

during development. The *Function Specification* defines the complete external behavior of the software, which is closely coupled to user interfaces.

Document the software boundaries and external stimuli and responses in the *Function Specification*.

Task 3

Specify the required external behavior of the software in the black box function form of stimulus history mappings to corresponding responses.

Specify the required external behavior of the software in all possible circumstances of use.

EXPLANATION: "All possible circumstances of use"

The *Function Specification* defines the required behavior of the software for all usage, including correct and incorrect, frequent and infrequent, and nominal and stress conditions. Responses for all possible stimulus histories should be specified.

EXPLANATION: Mathematical function

"Function" refers to a mathematical function. A mathematical function defines a mapping from a domain to a range. In a black box specification, the domain is the set of all possible sequences of inputs (all stimulus histories), and the range is the set of all correct responses. A mathematically "complete" specification is one in which all possible stimulus histories have been mapped to their corresponding responses. A mathematically "consistent" specification is one in which no history has been mapped to more than one response or one set of responses. A "correct" specification is one in which the domain, range, and mapping have been properly specified in the judgment of domain experts.

REFERENCE: Software specification based on mathematical function theory

[Mills 86], [Mills 87], [Mills 88]

RECOMMENDATION: Prudent exceptions to black box specification

The black box specification has a state- and procedure-free form that is extremely useful for validating requirements and driving incremental development and certification. In some cases, however, state box and even clear box specifications can be considered when they are more natural alternatives. Black box specifications are generally best, and well worth the effort to develop.

Use abstractions such as specification functions in the black box specification to maximize understandability, limit complexity, and maintain intellectual control.

EXPLANATION: Specification functions

"Specification functions" are a common form of abstraction used in scaling up black box specifications for large systems. Specification functions define conditions or operations that are used to simplify function mappings. They appear in the mappings as named placeholders. For example, in a specification for a database, a specification function named "delete-ok" operating on stimulus history might define conditions for which a "delete" stimulus should produce a deletion, namely, that the record to be deleted had been added somewhere in the history of use and not subsequently deleted.

Simplify external software behavior wherever possible to improve usability and reduce development and certification effort.

Document the black box mapping of stimulus histories to responses, and associated assumptions, in the *Function Specification*.

Verification

Verification 1 **Verify the completeness, consistency, correctness, and clarity of the evolving *Function Specification* work product in frequent team reviews.**

Verification 2 **Verify the completed *Function Specification* work product with the customer and the project team.**

Review the *Function Specification* with the customer, the development and certification teams, and affected peer groups for agreement on the basis for incremental development and certification.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria are satisfied.

The *Function Specification* has been completed, verified against *Software Requirements*, and agreed to by the customer as the basis for software development.

Usage Specification Process

A Cleanroom specification process

The purpose of the Usage Specification Process is to 1) identify and classify software users, usage scenarios, and environments of use, 2) establish and analyze the highest level structure and probability distribution for software usage models, and 3) obtain agreement with the customer on the specified usage as the basis for software certification.

The specification team creates the *Usage Specification* based on the *Software Requirements* and the evolving *Function Specification.* The information in the *Usage Specification* defines the scope of the testing effort and serves as the basis for incremental usage model development. It also assists in completing and validating the *Function Specification.*

Analysis of high-level usage models provides early guidance for allocation of development and testing resources. The analysis can provide estimates of relative long run usage of specified software functions, which can help prioritize development activities. It can also help estimate testing resource and schedule requirements. Usage model analysis can be carried out prior to software development, before resources are committed to increment development and certification. This analysis is an effective management tool for reducing the risk of inaccurate resource and schedule estimates.

Objectives

Objective 1 **The users, usage scenarios, and usage environments of the software are defined and documented to clarify the specification, inform development priorities, and provide a basis for initial test planning.**

Objective 2 **The customer agrees with the usage specification as the basis for usage model development and software certification.**

Participants

Specification team and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 The *Software Requirements* and the *Function Specification* have been partially or fully completed.

> EXPLANATION: Incremental usage specification
>
> All software requirements and function specifications must eventually be defined to permit complete usage specification. Often, all requirements are not fully understood at the outset, and a strategy of incremental usage specification based on partial requirements definition may be necessary. In large-scale developments, incremental usage specification is often a desirable strategy for pacing development, maintaining intellectual control, and eliciting customer feedback.

Entry 2 The *Usage Specification* requires revision for changes to the *Software Requirements* or *Function Specification*, or for changes from increment specification, development, or certification.

Entry work products and these supporting work products are available.

The *Engineering Change Log* describes proposed changes.

Tasks

Task 1

Define the format and notation of the *Usage Specification.*

The *Usage Specification* is often represented as a high-level Markov chain. Naming and documentation conventions are established for encoding usage information as elements of the chain.

> EXPLANATION: Markov chain
>
> Software use is a stochastic process that can be described as a Markov chain. A Markov chain can be represented as a directed graph, where the nodes are states of use and the arcs are stimuli that cause transitions between states.
>
> In the Usage Modeling and Test Planning Process, the high-level *Usage Specification* is refined to produce detailed Markov chain *Usage Models*. Additional explanation of Markov chains is given in that process.

Task 2

Specify the expected usage of the software through progressive stratification of usage characteristics.

> EXPLANATION: Stratification of usage characteristics
>
> Variation in usage can be described as a hierarchy of progressively narrower categories of usage. A heterogeneous user population, for example, may be subdivided into a set of more homogeneous user classes. This "stratification of usage" results in a better understanding of software usage requirements and provides a high-level basis for test planning.

Identify and classify all hardware, software, and human users of the software.

Identify the expected proportion of each class of user within the set of expected users.

EXAMPLE: User classifications

Hardware user classifications include sensors, actuators, and other peripheral devices. Software user classifications include operating systems, databases, and other controlling or supporting software. Human user classifications include job type, access privileges, and experience level.

EXPLANATION: Contribution of usage specification to function specification

Identification of users, usage scenarios, and environments of use in the Usage Specification Process contributes to the completeness and correctness of function definition in the Function Specification Process.

The users of the software are the sources of stimuli and the targets of responses. The completeness of the set of identified users is a necessary condition for the correctness of the domain defined in the *Function Specification.* The principle of "transaction closure" in Cleanroom black box specification refers to the requirement that all possible uses by all possible users be identified.

For each class of user, identify and classify all scenarios of use, including starting and ending events.

Identify the expected proportion of each class of scenarios within the set of expected scenarios.

EXAMPLE: Use classifications

Usage scenarios are defined by considering main and supporting user functions, routine and non-routine use, safe

and hazardous use, and other dimensions that stratify and organize usage patterns.

Since statistical testing is based on random sampling of the population of possible uses, the definition of a "use" is critical to the validity of the testing process. A "use" begins and ends with predefined events that are appropriate to the application, for example, invocation to termination, switchhook up to switchhook down, power up to power down, main menu to main menu, transaction start to transaction end, etc.

For each class of user and class of use, identify and classify expected hardware and software environments for the software system.

Identify the expected proportion of each class of environment within the set of expected environments.

EXAMPLE: Environment classifications

Usage environments can be classified in terms of characteristics such as computer and network configuration, capacity, and performance; system and support software capabilities and resource requirements; data rates and volumes; and support for concurrency.

EXPLANATION: Operational use as the context for certification

Cleanroom testing is performed as a statistical experiment in which tested use of the software should reflect operational use to the greatest extent possible. Careful characterization of operational environments permits their accurate simulation in testing, which in turn permits valid estimates of fitness for use of the software in the operational environments.

REFERENCE: Usage specification

[Walton 95a]

Document the results in the *Usage Specification.*

Task 3 **Represent usage information as high-level Markov chains. Analyze the models, and make recommendations based on analysis of usage model statistics.**

> EXPLANATION: Relationship of usage specification to usage modeling
>
> Usage specification is a system-level activity; detailed usage modeling parallels lower-level development activity. The high-level Markov chain developed during the Usage Specification Process is the top level of the usage model(s) developed during the Usage Modeling and Test Planning Process.

Identify any areas where the functions defined in the *Function Specification* result in excessive complexity and cost in usage model development. Make recommendations for possible simplification.

Evaluate software functions in terms of probability of use. Make recommendations on development priorities.

Analyze usage statistics to estimate resources and schedules required to achieve certification goals.

Verification

Verification 1 **Verify the evolving *Usage Specification* work product in specification team reviews.**

Conduct frequent specification team reviews of the evolving *Usage Specification* for completeness, consistency, correctness, and clarity.

Verification 2 **Verify the completed *Usage Specification* work product with the customer and the project team.**

Review the *Usage Specification* with the customer, the certification team, and affected peer groups for agreement on the basis for usage model development and software certification.

Measurement

Measurement 1 **Apply standard calculations to Markov chain usage models to derive high-level operational profiles of the software.**

> EXPLANATION: Usage model calculations
>
> Standard calculations on Markov chain usage models provide estimates of long-term software usage behavior. The calculations may be interpreted to identify patterns of use, usage features, probabilities of particular usage events, and insights relevant to both development and test planning. Further discussion of Markov usage model analysis is given in the Usage Modeling and Test Planning Process.

Exit

The process is complete when the exit criteria are satisfied.

The *Usage Specification* has been completed, verified, and agreed to by the customer as the basis for detailed usage modeling and test planning.

Architecture Specification Process

A Cleanroom specification process

The purpose of the Architecture Specification Process is to define the conceptual model, the structural organization, and the execution characteristics of the software. Architecture definition is a multi-level activity that spans the life cycle. Architecture may be inherited from a domain or product line, evolve within the constraints of the system of which it is a part, or wholly originate in the software project.

The Cleanroom aspect of architecture specification is in decomposition of the history-based black box *Function Specification* into state-based state box and procedure-based clear box descriptions. This high-level box structure of the software identifies and connects principal components, including their state encapsulations and operations. It is the beginning of a referentially transparent decomposition of the *Function Specification* into a box structure hierarchy, and will be used during increment development. The architecture may take a variety of forms, including functional, object-based, etc.

Key dimensions of architecture are 1) conceptual architecture expressed in terms of major software components and their relationships, 2) module architecture expressed terms of layers of functional decomposition, and 3) execution architecture expressed in terms of dynamic software operation [Soni 95]. The architecture is a vehicle for incorporating existing reference models, components, protocols, standards, and software design strategies. Architecture specification spans the development life cycle.

The *Software Architecture* is a principal input to the Increment Planning and Increment Design Processes.

Objectives

Objective 1	**The architectural strategy leverages existing assets and supports reuse plans.**
Objective 2	**The architectural structure of the software is defined as the complete behavior and interaction of its principal components.**
Objective 3	**The customer agrees with the software architecture as the basis for software development.**

Participants

Specification team, the development team, and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — The *Software Requirements* and the *Function Specification* are partially or fully completed.

Entry 2 — The *Software Architecture* requires revision for changes to the *Software Requirements* or *Function Specification*, or for changes from increment specification, development, or certification.

Entry work products and these supporting work products are available.

The *Engineering Change Log* describes proposed changes. The *Usage Specification* is used to clarify requirements and constraints on the software architecture.

The *Reengineered Software*, if any, is used to define use of reengineered components in the architecture.

Tasks

Task 1 **Identify architectural assets.**

Identify and analyze architectural assets applicable to the software, including existing domain models, reference architectures, components, communication protocols, standards, and design strategies and conventions.

Document the asset analysis in the *Software Architecture.*

Task 2 **Define a strategy for the software architecture.**

Define a strategy for the architecture based on the *Software Requirements, Function Specification,* the analysis of architectural assets, and requirements derived from higher-level system or subsystem design.

Document the strategy in the *Software Architecture.*

Task 3 **Specify the top-level box structure of the software architecture.**

Decompose the history-based black box specification of required external behavior defined in the *Function Specification* into top-level state-based state box and procedure-based clear box forms based on the architecture strategy.

For the state box, invent principal state elements and operations required to achieve specified black box behavior.

For the clear box, invent procedures for operations on state elements required to achieve specified state box behavior. Within the clear box, invent and connect principal software components, usually defined as black boxes, whose subsequent state box decompositions will encapsulate state at the next level.

Continue the decomposition until the architecture is fully elaborated.

The completed software architecture represents a hierarchy of box uses, wherein every use of a box is explicitly represented in the hierarchy.

Document the architecture in the *Software Architecture*.

Task 4 **Analyze and validate the software architecture.**

Perform simulations and analysis as necessary to ensure that performance, reliability, usability, and other software requirements can be met by the architecture.

Document the analysis in the *Software Architecture*.

Verification

Verification 1 **Verify the evolving *Software Architecture* work product in team reviews.**

Conduct frequent team reviews of the evolving *Software Architecture* to ensure that it meets requirements.

Use the Correctness Verification Process to verify that the representation of the *Software Architecture* in top-level box structure form is complete, consistent, and correct.

Verification 2 **Verify the completed *Software Architecture* work product with the customer and the project team.**

Review the *Software Architecture* with the customer, the development and certification teams, and affected peer groups for agreement on the basis for incremental development and certification.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria are satisfied.

The *Software Architecture* has been completed, verified, and agreed to by the customer.

Increment P nning Process

A Cleanroom specification process

The purpose of the Increment Planning Process is to 1) allocate customer requirements defined in the *Function Specification* to a series of software increments that satisfy the *Software Architecture*, 2) define schedule and resource allocations for increment development and certification, and 3) obtain agreement with the customer on the increment plan.

The *Increment Construction Plan* is created by the specification team for use by management to assign tasks, track progress, and monitor product quality and process control in the Project Management Process. It is revised as necessary to incorporate changes or accommodate actual project performance. In the incremental process, a software system grows from initial to final form through a series of increments that implement user function, execute in the system environment, and accumulate into the final system. The first increment is an "end-to-end" (i.e., initial user state to final user state) executable subset of the functional behavior on which later increments can build. When the final increment is in place, the system is complete. By providing a series of accumulating subsets of the software that grow in capability, the incremental process reduces risk and permits early and continual user evaluation and feedback. If the customer prefers delivery of the final system only, incremental development can still be used by the development organization for management control, risk mitigation, and to support development needs such as hardware-software co-design.

Incremental development and certification avoids the risks associated with a separate integration step late in a project life cycle. Increments are typically developed in top-down fashion, often with concurrent engineering of increments. Each increment is a decomposition of functions and interfaces specified in prior increments. This approach permits continual testing and quality assessment as the software evolves into final form.

Objectives

Objective 1 — **The incremental development and certification plan supports intellectual control of the work, statistical quality control of the process, and risk management of the overall project.**

Objective 2 — **The increment plan ensures ongoing clarification of requirements through user execution and evaluation of increments.**

Objective 3 — **The customer agrees with the increment plan as the basis for software development and certification.**

Participants

Specification team, chief development engineer, chief certification engineer, project software manager, and the customer.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — The *Software Requirements, Function Specification, Usage Specification, Software Architecture, Reuse Analysis Plan, Risk Analysis Plan,* and *Schedule and Resource Plan* are partially or fully completed. These work products are the basis for developing the *Increment Construction Plan,* as well as a source of revisions to it.

Entry 2 — The *Increment Construction Plan* requires revision for changes from development or certification activity or as a result of new or changed requirements.

Entry work products are available.

Tasks

Task 1

Partition software functions into a series of increments for development and certification.

Define the functional content of a series of software increments that implement user function, execute in the system environment, and accumulate into the final system.

> EXPLANATION: "Accumulate into the final system"
>
> Cleanroom increments accumulate in a top down fashion. The *Function Specification* and *Software Architecture* provide the high-level structure for a series of increments that grow from the structure, each increment a further decomposition of the previous one. From the beginning, embedded specifications with executable "stubs" are used as placeholders for functions planned for later increments. In this way, all testing occurs in a system environment; traditional integration testing is unnecessary.

Use the *Software Requirements* to identify software requirements or system engineering factors that may influence the definition of increment content.

Use the *Function Specification* and the *Software Architecture* to identify required software functions and their dependency relationships as a basis for defining increment content.

Use the *Reuse Analysis Plan* to identify reused components and allocate them to appropriate increments.

Use the *Risk Analysis Plan* to identify risks that influence increment content. Plan increment content to avoid or manage risks, with emphasis on addressing risks early in the project.

Use the *Usage Specification* to define increment content in consideration of usage probabilities, specifically, to

incorporate functions with high usage probabilities into early increments.

Identify special components, such as complex algorithms requiring extensive analysis, for independent development and certification prior to incorporation into the accumulating increments. These components can be incorporated as reusable assets in the overall increment plan.

Document the required functional content of the increments in the *Increment Construction Plan.*

Task 2

Refine the *Schedule and Resource Plan* by allocating schedules and resources to increment development and certification.

Within the overall constraints of the *Schedule and Resource Plan,* allocate development and certification schedules and resources for each increment.

Provide for overlapping or parallel development of increments as necessary to meet schedules based on availability of development resources.

Define schedule points for measurement of software quality and process control, and for customer evaluation of increments.
Document schedule and resource allocations in the *Increment Construction Plan.*

REFERENCE: Increment planning [Trammell 96]

Verification

Verification 1

Review the *Increment Construction Plan* with the customer, the development and certification teams, and affected peer groups for agreement on the basis for increment development and certification.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criteria is satisfied.

The *Increment Construction Plan* has been completed, verified, and agreed to by the customer as the plan for software development and certification.

Software Reengineering Process

A Cleanroom development process

The purpose of the Software Reengineering Process is to prepare reused software for incorporation into the software product. Reused software can originate in Cleanroom or non-Cleanroom environments, and can include commercial products, customer-furnished software, and components from previous software developments. Software may be reused as is, reused through "firewalls" that shield Cleanroom software, or reused after reengineering.

Reused software must satisfy two principal Cleanroom requirements. First, the functional semantics and interface syntax of reused software must be understood and documented, to maintain intellectual control and avoid unforeseen failures in execution. If specification and design documentation for reused software is incomplete, its functional semantics can be recovered through function abstraction and correctness verification. The completeness and correctness of specifications for reused software must satisfy project specification standards.

Second, the fitness for use of reused software must be either known or determined, in order to achieve the project's certification goals. Usage models can be developed for reused software, and its fitness for use determined through statistical testing. The reliability of reused software must satisfy project certification goals.

The results of the Software Reengineering Process are documented in the *Reengineering Plan* and *Reengineered Software.*

Objectives

Objective 1	**Reengineered software satisfies requirements for the software product in which it is used.**
Objective 2	**Reengineering activity enables intellectual control over the reengineered software.**
Objective 3	**Reengineered software is certified to be fit for its intended use as necessary to meet certification goals for the software product in which it is used.**

Participants

Development team, specification team, and certification team.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — Candidate reusable assets identified in the *Reuse Analysis Plan* are to be evaluated and possibly reengineered for use in the software product.

Entry 2 — The *Reengineering Plan* and/or the *Reengineered Software* require revision for changes from specification, development, or certification activities.

Entry work products and these supporting work products are available.

Reused software and its supporting documentation are used as the basis for creating the *Reengineered Software.*

The *Engineering Change Log* describes proposed changes. The *Software Requirements, Function Specification, Usage Specification, Software Architecture, and Increment*

Construction Plan are used to define requirements for reengineering reused software.

Tasks

Task 1 **Analyze candidate reused software and its documentation to develop a reengineering plan.**

Analyze specifications, designs, and implementations of reused software to evaluate the completeness and correctness of documentation of its functional semantics, and the extent of reeengineering necessary to satisfy software product requirements.

Analyze the usage models, test plans, test procedures, test results, and actual usage of reused software to evaluate the basis for its reliability estimates.

Conduct a cost/benefit analysis with respect to project certification goals and future software maintenance responsibilities to determine appropriate resource allocations to reengineering activities.

If necessary, develop a plan for reengineering reused software to satisfy functional requirements, recover functional semantics, and/or assess fitness for use.

Define and document reengineering tasks, schedules, and resources in the *Reengineering Plan.*

Task 2 **Recover the functional semantics of reused software using function abstraction techniques.**

If reused software implementations are not structured, transform them into structured form using program structuring techniques to permit function abstraction.

Carry out stepwise abstraction of structured implementations as necessary and document embedded intended functions. Continue abstraction until

specifications of external behavior in all possible circumstances of use have been defined.

EXAMPLE: Recovering a specification through function abstraction

Consider the following miniature program, which has no intended functions.

```
do
   if y < 0
   then t := -y
   else t := y
   endif;
   if x > t
   then z := x
   else z := t
   endif
enddo
```

The first ifthenelse can be read and abstracted to [t := abs(y)], and the second to [z := max(x, t)], to produce an intermediate level of design:

```
do
   [t := abs(y)];
   [z := max(x, t)]
enddo
```

This sequence program can in turn be abstracted to obtain the overall specification of the program:

```
do
   [z := max(x, abs(y))]
enddo
```

The abstractions can be recorded in the program text to document intermediate and final specifications.

Document the functional semantics of reused software in *Reengineered Software.*

Task 3 **Reengineer reused software to meet software product requirements.**

Respecify, redesign, and reimplement reused software as necessary to meet requirements, using the Function Specification, Increment Design, and Correctness Verification Processes.

Document the reengineering of reused software in *Reengineered Software.*

Task 4 **Recover the functional semantics of reused software using experimental execution.**

If the source code of reused software is not available, conduct experimental executions as necessary to derive an understanding of its functional semantics.

> RECOMMENDATION: Use of COTS or APIs
>
> If neither specifications nor source code are available, execution experiments can be used to understand the semantics of the software. The use of the COTS, API, or otherwise "sealed" software in the product under development should be restricted to functions that are well understood.

Document the functional semantics of reused software in *Reengineered Software.*

Task 5 **Certify the fitness for use of reused software.**

Create usage models and conduct statistical testing as necessary to certify the fitness for use of reused software with respect to project certification goals. Use the Usage Modeling and Test Planning Process and the Statistical Testing and Certification Process.

Document certification results in *Reengineered Software.*

Verification

Verification 1 **Verify the *Reengineered Software* work product.**

Carry out correctness verification in team reviews as necessary to ensure correctness of abstracted specifications and/or redeveloped software. Use the Correctness Verification Process.

Measurement

Measurement 1 Measure the fitness for use of the reengineered software using the Cleanroom certification processes and associated measures.

Exit

The process is complete when one of the exit criteria is satisfied.

Exit 1 The *Reengineered Software* has been completed, including any necessary redevelopment to meet requirements, abstraction of functional semantics, and certification of fitness for use.

Exit 2 Reengineering activity has revealed that the candidate software is not fit for use in the product and project plans must be changed.

Increment Design Process

A Cleanroom development process

The purpose of the Increment Design Process is to design and code a software increment that 1) satisfies the *Increment Construction Plan, Function Specification,* and *Software Architecture,* and 2) conforms to Cleanroom design principles and quality criteria. The development team documents each increment in the *Increment Design.*

Increments are designed and implemented as usage hierarchies through box structure decomposition. This process preserves referential transparency between successive decompositions to maintain intellectual control. Increment designs can be expressed in object, functional, or other forms. Each increment is based on a prior specification. Increment specifications are expressed in stimulus history-based black box and state-based state box forms. Increment designs and implementations are expressed in procedure-based clear box forms that can introduce new black boxes for further decomposition. Reused or reengineered components are incorporated as planned.

Team reviews during the Increment Design Process focus on issues such as clarity, maintainability, reuse, and conformance to style. In the companion Correctness Verification Process, the team focuses exclusively on correctness. Specifications, designs, and implementations evolve during the Increment Design Process, and intended functions are embedded in clear box procedure decompositions to permit effective correctness verification. The team performs correctness verification as the last intellectual pass through the work.

The development team does not execute the increment implementation. First execution is performed by the certification team in the Statistical Testing and Certification Process after the development team has completed verification in the Correctness Verification Process.

Objectives

Objective 1 — **The increment design and implementation satisfy the *Function Specification*, the *Software Architecture*, and the *Increment Construction Plan*.**

Objective 2 — **The increment design and implementation are verifiably correct decomposition of required functions.**

Objective 3 — **Intellectual control over increment design and implementation is maintained through team reviews.**

Participants

Development team, with specification team available for consultation and review.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — The *Software Requirements, Function Specification, Usage Specification, Software Architecture, Reengineered Software,* and *Increment Construction Plan* are sufficient for increment design, and a software increment is scheduled for development or change. These work products are the basis for developing the *Increment Design,* as well as a source of revisions to it.

Entry 2 — An *Increment Verification Report* or *Increment Certification Report* identifies faults or failures requiring correction of the *Increment Design.*

Entry work products and the supporting work product are available.

The *Engineering Change Log* describes proposed changes.

Tasks

Task 1 **Review the work products that are the basis for the increment design.**

Review the *Increment Construction Plan* to identify the user functions to be implemented in the increment.

Review the *Function Specification* for definitions of the user functions to be implemented in the increment.

Review the *Software Architecture* for the architectural strategy to be maintained in the increment.

Task 2 **Design and implement the software increment as a usage hierarchy through box structure decomposition.**

> EXPLANATION: Box structure usage hierarchy
>
> Box structure decomposition results in a usage hierarchy of objects, modules, and other units of code. The box structure hierarchy for an increment is the completed increment; the box structure hierarchy for the final increment is the completed software product.

Decompose history-based black box specifications into state-based state box specifications with equivalent behavior in all circumstances of use.

Decompose state box specifications into procedure-based clear box designs with equivalent behavior in all circumstances of use. Introduce new black box uses in clear box designs as necessary.

Create clear box designs as structured procedures that fully define control and data relationships among new black box uses and other design elements.

Repeatedly decompose new black boxes into state box and clear box forms. Continue decomposition until designs can be implemented with no further invention required.

Maintain referential transparency between decompositions for intellectual control.

> EXPLANATION: Referential transparency
>
> Cleanroom minimizes the risk of integration faults through development based on the mathematical principle of referential transparency. Referential transparency in box structure hierarchies requires that the black box specifications embedded in clear boxes at each level of decomposition precisely define the required functional behavior of their subsequent decompositions into state and clear boxes. With referential transparency, intellectual control is maintained and independent work at lower levels can proceed without concern for functional interactions at higher levels.

Incorporate components from *Reengineered Software* into the increment as planned.

Attach intended functions to the control structures in clear box procedure designs for use in correctness verification.

> EXPLANATION: Intended functions
>
> Intended functions are a key Cleanroom concept and are essential to achieving Cleanroom objectives. An intended function is a definition of the full functional effect on data of the control structure (sequence, ifthenelse, whiledo, etc.) to which it is attached. Intended functions typically appear as comments in the clear box. They are often expressed in black box or state box form, particularly as conditional rules, and are used in verifying their control structure expansions.

EXAMPLE: Intended function

The following ifthenelse operates on integers. Its intended function is attached in square brackets:

```
[set z to maximum of x and y]
if x > y
then  z := x
else  z := y
endif
```

The intended function and its control structure decomposition are referentially transparent. The ifthenelse is correct if it carries out the operations on data prescribed by its intended function.

If necessary, translate designs into the implementation language and review for correct translation.

Refer to the *Usage Specification* for information about the operational environment. Refer to the *Increment Verification Report* or *Increment Certification Report* for faults or failures requiring correction.

Document the design and code in the *Increment Design.*

REFERENCE: Box structure specification and design

[Mills 86], [Mills 87], [Mills 88]

Task 3

Improve the *Increment Design* through team reviews.

Conduct frequent development team reviews of the evolving *Increment Design* to discuss design strategies and improvements, and assess characteristics including understandability, verifiability, and maintainability. Make design simplification and style compliance explicit review objectives for efficient correctness verification. Redesign for simplicity where cost effective.

EXPLANATION: Writing for verification

Correctness verification is only possible if designs are verifiable. This is not to say that designs are not *correct* unless they are verifiable, only that they are not *verifiably* correct.

Cleanroom designs are written for verification. The stepwise unfolding of specification and design in box structure decompositions ensures traceability of design to specification at every level of the usage hierarchy. Each specification is "distributed" as intended functions for design components during the Increment Design Process, and design components are verified against their intended functions during the Correctness Verification Process.

Identify opportunities for state migration and use of common services.

EXPLANATION: State migration

State migration is a Cleanroom strategy for improving and simplifying designs.

State migration concerns placement of state data at the most effective level of decomposition for its use. It implements the software engineering principle of information hiding for limitation of data scope. State migration places data based on its scope of usage at as low a level in a system hierarchy as possible, but at as high a level as is necessary. Migration of state data may be possible whenever new black boxes are created in a given clear box. Any state data item used solely by one lower level box can be migrated to it.

EXPLANATION: Common services

Use of common services is another Cleanroom strategy for improving and simplifying designs.

Common services are reusable components. They may be newly created for a given system, or drawn from a reuse

library. Common services afford economy in system size, effective use of development resources, efficient verification, and increased reliability.

Task 4 **Perform individual correctness verification.**

Apply function-theoretic correctness verification on an individual basis to evolving designs, with the objective of entering the Correctness Verification Process with few faults.

Verification

Verification of the correctness of the *Increment Design* is so critical to Cleanroom objectives that an entire process is devoted to it. See the Correctness Verification Process.

Measurement

See Common Cleanroom Process Elements on page 21.

Exit

The process is complete when the exit criterion is satisfied.

The *Increment Design* has been completed.

Correctness Verification Process

A Cleanroom development process

The purpose of the Correctness Verification Process is to verify the correctness of the software increment using mathematically based techniques.

Correctness verification is carried out in development team reviews using function-theoretic reasoning. Black box specifications are verified to be complete, consistent, and correct. State box specifications are verified with respect to corresponding black box specifications. Clear box procedures are verified with respect to corresponding state box specifications; every control structure in every clear box procedure is verified against its intended function using the Correctness Conditions of the Correctness Theorem [Linger 79]. Faults found in verification reviews are documented in the *Increment Verification Report* and are corrected by the specification and development teams under engineering change control. The specifications and designs are then reverified.

Written proofs of correctness based on function-theoretic techniques provide additional rigor if necessary for life-, mission- and enterprise-critical software.

The Correctness Verification Process is concurrent with the Increment Design Process. Correctness verification is the last intellectual pass at each level of decomposition, the last line of defense against failures in statistical testing and certification. The objective of correctness verification is to enter testing with no faults in the implemented design. Following completion of verification by the development team, the increment is turned over to the certification team for first execution.

Objectives

Objective 1 **The team agrees that the software increment is correct with respect to its specification, i.e., that it contains no remaining faults.**

Objective 2 **Faults and inadequacies found in correctness verification are documented to permit subsequent analysis for process improvement.**

> REFERENCE: CMM Defect Prevention KPA
>
> If compliance with this KPA is an organizational objective, its specific requirements should be reviewed when the Correctness Verification Process is tailored for organizational or project use.

Participants

Development team, with specification team available for consultation.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 A new *Increment Design* has been completed or is in progress.

Entry 2 A reengineered or corrected *Increment Design* has been completed or is in progress.

Entry work products and these supporting work products are available.

The *Function Specification* defines the required external behavior of the functions allocated to the increment in the *Increment Construction Plan.*

The *Software Architecture* defines the architectural strategy to be used in the increment design.

Tasks

Task 1

Verify the correctness of the software increment using mathematically-based verification techniques.

Verify the correctness of every specification and design structure in the *Increment Design.*

Carry out verbal proofs of correctness based on function-theoretic techniques in team verification reviews. A consensus of team members is required to establish correctness.

For black box verification, determine the completeness, consistency, and correctness of its specification.

For state box verification, compare state box behavior to corresponding black box behavior for equivalence.

For clear box verification, apply the Correctness Conditions of the Correctness Theorem to determine the correctness of every control structure (including embedded black box specifications) with respect to its intended function.

EXPLANATION: Correctness conditions for basic control structures

"Correctness Conditions" for the three basic control structures are given as follows in question form. (f stands for intended function, p for predicate, g and h for operations on data.)

Sequence:

[f] do	Does g followed by h do f?

```
    g;
    h
  enddo
```

Ifthenelse:

```
  [f]                    When p is true
  if p                     does g do f,
  then                   and when p is false
    g                      does h do f?
  else
    h
  endif
```

Whiledo:

```
  [f]                    Does the loop terminate,
  while p                and when p is true
    g                      does g followed by f do f,
  enddo                  and when p is false
                           does doing nothing do f?
```

EXAMPLE: Correctness verification

The following miniature program that operates on integers contains three control structures, namely, two ifthenelses in a sequence (max is short for maximum, abs for absolute value).

```
do [z := max(x, abs(y)) and t := abs(y)]
  [t := abs(y)]
  if  y < 0
   then
     t := -y
  else
     t := y
  endif;
  [z := max(x, t)]
  if x > t
   then
     z := x
```

```
   else
     z := t
   endif
enddo
```

For verification, the Correctness Conditions corresponding to each control structure must be checked:

First ifthenelse:

When y < 0 is true,
  does t := -y do [t := abs(y)],
and when y < 0 is false,
  does t := y do [t := abs(y)]?

Second ifthenelse:

When x > t is true,
  does z := x do [z := max(x, t)],
and when x > t is false,
  does z := t do [z := max(x, t)]?

Sequence:

Does [t := abs(y)] followed by [z := max(x, t)] do [z := max(x, abs(y) and t := abs(y)]?

In each case, the answer is yes. Note that in the final sequence verification, the ifthenelse logic does not appear, having been replaced by its equivalent intended function definitions. Thus, the verification of individual control structures does not depend on their decompositions, which are verified separately. The reasoning required in each verification step is thereby localized and kept manageable to help preserve intellectual control, even at high levels of design. In this way, early increments of systems can be completely verified for correctness, even though subsequent increments are yet to be designed.

RECOMMENDATION: Correctness conditions for other recurring constructs

Modern software development environments employ a wide variety of features and constructs that reduce to—but are often not easily recognizable as—standard control structures such as those mentioned above. Visual programming languages have graphical elements. Real-time facilities have timing mechanisms such as process rendezvous. Application generators have high-level resources such as GUI builders. Multi-tasking, multi-user, multi-threaded applications use such control mechanisms as resource locking. And so on.

A project team should establish the correctness conditions for such recurring constructs using function-theoretic reasoning. The development of standard verification protocols for recurring idioms or patterns is precisely the sort of process tailoring that needs to be done to adapt the Cleanroom process to a given project and environment.

Task 2 **Document findings of team verification reviews.**

Create an *Increment Verification Report* documenting all faults, problems, and improvements identified in verification reviews, and assign corrective actions.

Task 3 **Create written proofs of correctness as necessary for critical software.**

Develop written proofs of correctness as necessary for life-, mission-, and enterprise-critical software, and verify the proofs in team reviews.

Document the proofs in the *Increment Design.*

Task 4 **Reverify all corrections to faults.**

Perform reverification reviews on corrections to faults, including reverification of the full context of corrections to avoid unforeseen side effects.

Verification

Verification 1 **Confirm that every box structure has been verified as correct by team consensus.**

Confirm that every black box, state box, and clear box in the new and changed portions of the *Increment Design* has been verified to be correct.

Measurement

Measurement 1 **Measure the *Increment Design* and the Increment Design Process.**

The Correctness Verification Process is a focused team review of the *Increment Design*. Measure the quality of the *Increment Design* and the effectiveness of the Increment Design Process in terms such as the number, type, and severity of faults found in the verification reviews.

Exit

The process is complete when one of the exit criteria is satisfied.

Exit 1 The increment has been verified with no faults found.

Exit 2 The increment has been verified and contains faults that must be corrected and the engineering changes verified.

Exit 3 The black box, state box, or intended function definitions are insufficient for effective verification, and must be revised before verification can be accomplished.

Exit 4 Initial verification has found faults in sufficient quantity and severity that the process must be terminated and the increment redesigned.

In each case, the *Increment Verification Report* is created. Written proofs, if any, are added to the *Increment Design*. The Correctness Verification Process cannot be completed until the *Increment Design* is completed.

Usage Modeling and Test Planning Process

A Cleanroom certification process

The purpose of the Usage Modeling and Test Planning Process is to 1) refine the *Usage Specification* to create usage models for software testing, 2) define test plans, 3) obtain customer agreement on the usage models and test plans as the basis for software certification, and 4) generate statistical test cases and prepare the test environment.

The certification team creates the *Usage Models* and *Increment Test Plan,* and generates the *Statistical Test Cases.* Usage models are used to generate statistical test cases and monitor the progress of testing in the Statistical Testing and Certification Process. A usage model for a software system represents an infinite population of possible uses. It consists of a structural component that defines possible traversals of states of use by users, together with a probability component that defines the likelihood that particular traversals will occur. In statistical testing, test cases are generated from the usage model based on its probability distribution. Multiple usage models may be required for multiple classes of users and environments. Models are developed incrementally in accordance with the *Increment Construction Plan,* and accumulate into final form in parallel with increment designs. The customer reviews the usage models, and agrees that they will generate all scenarios of use, are correctly weighted, and are appropriate for certification.

Usage model statistics provide a great deal of information about the testing effort that will be required to achieve certification goals given projected failure rates in testing. Usage model analysis provides a basis for test planning, and is an effective management tool for reducing the risk of inaccurate resource and schedule estimates.

Objectives

Objective 1	**Valid usage models are defined which represent all possible uses of the software under expected or other usage conditions.**
Objective 2	**A statistical testing plan based on the usage models is defined and validated through model analysis and simulation.**
Objective 3	**The customer agrees on the usage models and statistical test plan as the basis for software certification.**

Participants

Certification team and customer, with specification team available for consultation and review.

Entry

The process begins when one of the entry criteria is satisfied.

Entry 1 — The *Usage Specification, Function Specification,* and/or *Increment Construction Plan* have been completed or changed. They are the basis for developing the *Usage Models* and *Increment Test Plan,* as well as a source of revisions to them.

Entry 2 — The *Usage Models* or *Increment Test Plan* require revision for changes from increment development or certification.

Entry work products and these supporting work products are available.

The *Software Architecture* and *Reengineered Software* may also provide information for development of the *Usage Models.*

The *Engineering Change Log* describes proposed changes.

Tasks

Task 1 **Define the usage models to be developed.**

Use the *Usage Specification* to define the usage models to be elaborated, and the scope and purpose of each.

Include special purpose models as necessary, for example, for certification of infrequently used functions with high consequences of failure.

> EXPLANATION: Special purpose models
>
> A usage model represents the conditions under which software is used. In general, expected usage conditions are modeled. In addition, other usage conditions may be of interest as well, and are modeled for special purposes. Hazardous usage conditions, for example, may be of interest for safety critical software. Malicious usage conditions might be modeled for software with special security requirements. Usage can be characterized in whatever terms are important in the certification context.

Consider use of actual user input where available. Real-time data feeds or the output of automated usage capture facilities can be used as components of usage models.

Task 2 **Define the structure of each usage model.**

Refine the *Usage Specification* to develop the *Usage Models*. For each model, define all possible usage states and their transitions based on the functions required by the *Increment Construction Plan* and defined in the *Function Specification*.

Define the structure of each model in the *Usage Models* document.

RECOMMENDATION: Markov chain usage models

The structure of a usage model can be represented as a Markov chain. A Markov chain usage model reflects the stochastic nature of software use, and permits analysis of usage and automation of test activity.

The usage model structure represents all possible uses of the software, expressed in terms of the initial usage state, subsequent sequences of possible usage states, and the terminal usage state. The model can be represented as a directed graph, whose nodes (usage states) are connected by arcs (possible transitions in use). Any usage scenario can be generated from a traversal of the model structure.

Ambiguity, inconsistency, or complexity in the *Function Specification* is often identified during creation of usage model structures.

REFERENCE: Usage modeling

[Whittaker 93], [Whittaker 94a]

RECOMMENDATION: Early planning for test automation

It is crucial to anticipate test automation requirements during usage modeling. Linkage with test tools, pre- and post-processing steps, live data feeds, response capture facilities, and numerous other aspects of automated testing are likely to be simpler if test automation is considered during usage modeling.

Task 3 **Define the transition probabilities of each usage model.**

Determine transition probabilities between usage states based on usage information and certification goals.

Employ user estimates and experience with similar systems and prior versions as sources of information about usage probabilities.

Define transition probabilities for each model in the *Usage Models* document.

EXPLANATION: Transition probabilities

While the structure of the usage model defines possible use, the transition probabilities define expected use. The probabilities associated with the transitions in the usage model may be known, partially known, or unknown. If they are known, as is often the case with well-instrumented systems in mature domains, the probabilities can be directly assigned. If they are not known, they can be estimated or made uniform. If they are partially known, a combination of these strategies can be used.

Probabilities can also be defined for other than expected use, for example, to emphasize testing of infrequently used functions with high consequences of failure.

The validity of conclusions drawn in statistical testing is entirely related to the usage models employed. Systematic acquisition of knowledge about expected usage is essential for developing accurate usage models.

REFERENCE: Optimization of usage models

Cleanroom practice is evolving toward automatic generation of transition probabilities from usage constraints. Operations research techniques can be applied to optimize usage models for an objective function, such as minimum testing cost, subject to usage constraints that characterize available knowledge about expected use.

[Poore 95b], [Walton 95b]

Task 4 **Validate the usage models.**

Generate statistics for each usage model. Evaluate the statistics to validate the overall usage profile, and to estimate resources and schedules required to achieve certification goals.

Develop recommendations based on the analysis, for example, cost-saving simplifications to the user functions defined in the *Function Specification.*

EXPLANATION: Practical interpretation of usage model analysis

Important information is available through standard calculations on a Markov chain usage model, for example:

- the expected length of a usage scenario (i.e., test case length)
- the expected minimum number of usage scenarios until a given usage state occurs for the first time
- the expected occupancy of each state of use (as a proportion of all states of use) in the long-term use of the software
- the expected minimum number of test cases required to cover all states and all transitions of the model
- the expected number of test cases required to achieve target levels of reliability and confidence.

Interpretations of these calculations provide insights about potential hazards in use, allocation of development and testing resources, and other information for management decision making.

REFERENCE: Usage model analysis

[Whittaker 94b]

Task 5 **Develop a plan for certification testing of the software increment.**

Develop a test plan, including schedules, staffing, training, hardware and software environment, certification goals, use of statistical test cases, use of operational input, procedures for verifying correct software performance, and documentation.

Define the test plan to ensure experimental control, including test procedures, test monitoring, results recording, failure evaluation, and engineering change control.

EXPLANATION: Experimental control

Cleanroom testing is conducted as a statistical experiment to permit scientifically valid conclusions about the fitness for use of the software.

In a statistical experiment, a series of random trials is performed under specified conditions, the outcomes of the trials are determined according to specified criteria, and conclusions about the probabilities of the outcomes are drawn.

In statistical testing, the trials are test cases that are randomly generated from the usage models, the outcomes correspond to the performance of the software, and the conclusions concern the probabilities of correct and incorrect software performance. Conclusions are used to inform decisions about test management and product release.

Many aspects of statistical testing must be controlled to preserve the properties of the statistical experiment. Performing trials under specified conditions means, for example, that the same software version must be used in each test case; a new software version marks the beginning of a new experiment. Determining the outcomes of the trials according to specified criteria means, for example, that the judgments by the testers and the evaluations by the test oracles must be consistent across all test cases. Explicit policies and operating procedures are required to ensure experimental integrity in statistical testing.

REFERENCE: Experimental control

[Trammell 94], [Trammell 95]

Plan for additional testing techniques to be applied in conjunction with statistical testing as necessary.

EXPLANATION: Other testing strategies

Statistical testing for reliability certification is a form of random testing. Statistical methods for nonrandom testing are often used to accomplish specific objectives as well. Test cases producing the fastest coverage of the usage model, for example, might be generated for use at the beginning of testing to reveal any immediate problems with the software.

Some forms of nonstatistical testing may be included in the test plan as well, such as specific tests that are required by the customer, by a standard, or by law.

Document testing plans in the *Increment Test Plan.*

Task 6 **Generate the statistical test cases.**

Use the *Usage Models* to generate the *Statistical Test Cases* to be used in the statistical testing.

EXPLANATION: Manual vs. automated testing

For manual testing, the generated test cases might be "scripts" of instructions to human testers. For automated testing, the scripts might be command sequences.

Task 7 **Prepare the statistical testing environment.**

Establish the hardware configuration and software environment required to test the software.

EXPLANATION: Test environment

Preparation of the test environment may be a resource-intensive task. In such cases, it will receive special emphasis in the *Schedule and Resource Plan* developed during the Project Planning Process and in the *Usage Specification* developed in the Usage Specification Process.

Verification

Verification 1

Verify the evolving *Increment Test Plan* and *Usage Models* work products in team reviews.

Conduct frequent certification team reviews of the evolving *Increment Test Plan* and *Usage Models* for completeness, consistency, correctness, and simplicity. Confirm through quantitative analysis of usage model properties, such as the long run probabilities of state occurrence, that the models are consistent with user estimates and experience.

Verification 2

Verify the completed *Increment Test Plan* and *Usage Models* work products with the customer and the project team.

Review the *Increment Test Plan* and *Usage Models* with the customer, the specification and certification teams, and affected peer groups to obtain agreement on them as the basis for software certification.

Measurement

Measurement 1

Measure the *Usage Models* work product.

Measure the size of the *Usage Models* in terms such as the number of usage states, state transitions, and statistically typical paths.

Exit

The process is complete when the exit criterion is satisfied.

The *Increment Test Plan* and *Usage Models* have been completed and agreed to by the customer as the basis for software certification.

Statistical Testing and Certification Process

A Cleanroom certification process

The purpose of the Statistical Testing and Certification Process is to demonstrate the software's fitness for use in a formal statistical experiment. "Fitness for use" is defined with respect to the usage models and certification goals employed in the testing process. The certification goals, first established in the *Measurement Plan* and refined in the *Increment Test Plan*, may be expressed in terms such as software reliability, reliability growth rate, and coverage of the usage defined in the usage models.

Software increments undergo first execution in this process. The increments are compiled, the *Executable System* is built, the statistical test cases are executed under experimental control, and the test results are evaluated. The success or failure of test cases is determined by comparison of actual software behavior with the required behavior defined in the *Function Specification*. Failures found during statistical testing are documented in the *Statistical Testing Report*. Intermediate and final test results are evaluated to inform test management decisions. As testing proceeds, the values of certification measures are compared with certification goals. The results of the comparisons drive decisions on continuing testing, stopping testing for engineering changes, stopping testing for reengineering and reverification, and final software certification.

In addition to measuring software quality and reliability, certification metrics are also used as measures of process control. Cleanroom team performance standards based on historical data, such as failure rates in statistical testing of prior systems, are compared to current metrics to inform management decisions. Evaluations and decisions regarding product quality and process control are documented in the *Increment Certification Report*.

Objectives

Objective 1	**Software testing is conducted in a formal statistical design under experimental control.**
Objective 2	**The software is demonstrated to perform correctly with respect to its specification.**
Objective 3	**Statistically valid estimates of the properties addressed by the certification goals are derived for the software.**
Objective 4	**Management decisions on continuation of testing and certification of the software are based on statistical estimates of software quality.**

Participants

Certification team, development team, and project software manager.

Entry

The process begins when the entry criteria are satisfied.

The *Increment Test Plan* has been completed, the *Statistical Test Cases* have been generated, and the test environment has been prepared.

A new or corrected *Increment Design* is available for compilation.

The *Function Specification* and *Usage Models* are available for use in evaluating observed behavior against specified behavior.

Tasks

Task 1 **Prepare the software increment for testing.**

Compile the software increment. If corrections are necessary, initiate the Engineering Change Process. After successful compilation, create the *Executable System* work product containing the load modules required for execution.

Task 2 **Perform other types of testing if necessary.**

Perform other types of testing if necessary prior to statistical testing. For example, special testing may be required to demonstrate specific scenarios of use, or to achieve complete usage model coverage with the minimum number of test cases.

> EXPLANATION: Order of statistical and other testing
>
> The key consideration in determining whether to perform other types of tests before or after statistical testing is the effect on certification. When a reliability estimate is made at the conclusion of statistical testing, it applies to the specific version of the software that was tested. If changes are made as a result of subsequent testing, the reliability estimate no longer applies.
>
> It is generally preferable to perform any non-statistical tests prior to statistical testing. Non-statistical tests performed after statistical testing may invalidate the reliability certification if the software is changed.

Task 3 **Execute the statistical test cases in the test environment.**

Execute the *Statistical Test Cases* according to the procedures defined in the *Increment Test Plan.*

Task 4 **Evaluate the statistical test case results.**

Evaluate the correctness of the software responses with respect to the behavior defined in the *Function Specification.* If failures are observed, evaluate their impact on the continuation of testing, experimental control, and the validity of certification results. If corrections are necessary, initiate the Engineering Change Process.

> EXPLANATION: Independent trials
>
> A key requirement in a statistical experiment is that the trials be independent—that is, the outcome of one trial must have no effect on the outcome of any other trial. While randomly generated test cases may ensure independent trials in statistical testing, the requirement for independence can still be undermined by failures in testing. For example, if a failure on a test case "blocks" access to functions required by a subsequent test case, testing should be stopped the problem fixed.

Document test results in the *Statistical Testing Report.* Record data for each failure, including the test environment, test case, test results, and failure type and severity, together with any other information that will assist in determining its cause.

Task 5 **Derive certification measures.**

Use the *Usage Models, Statistical Test Cases, Statistical Testing Report,* and results of other testing to derive measures of the fitness for use of the software with respect to certification goals.

Measures can include reliability and confidence, reliability growth rate, mean time to failure, representativeness of the test case sample, and other measures derived from comparison of expected and observed software performance.

Use statistical methods such as hypothesis testing, interval analysis, and analysis of failure data with reliability models.

EXPLANATION: Reliability measurement

Software reliability measurement is a hallmark of Cleanroom. Reliability estimation based on Markov chain usage models is a prominent approach to reliability measurement in Cleanroom practice. The Markov chain approach provides measures of reliability, confidence, and other stopping criteria.

Classical statistical hypothesis testing is also used in Cleanroom for reliability estimation. Models of reliability growth can be used where their underlying assumptions are justified.

REFERENCE: Certification measures

[Whittaker 94b]

Document certification measures in the *Increment Certification Report.*

Task 6 **Compare certification measures with certification goals.**

Compare the values of trends in the certification measures with project goals for product quality and process control.

If appropriate, combine certification measures from the current statistical testing experiment with measures from other experiments.

EXPLANATION: Conditions for combining test information

If test conditions (for example, software version, usage model, execution environment) are the same, data from various statistical testing experiments can simply be combined. If testing conditions are not the same, more complex approaches to combining information must be used to ensure the validity of conclusions.

Document evaluations in the *Increment Certification Report.*

Task 7 **Decide whether or not to stop testing.**

Positive case: Testing can be stopped and the software certified as fit for use if 1) the values of the current certification measures satisfy certification goals, and 2) no failures have been observed during testing of the current software version (or none worth the cost and risk of correction).

Negative case: Testing should be stopped and the software reengineered and reverified process control standards have been violated. Violation of process control standards occurs when certification goals cannot be achieved given current values of the certification measures and the remaining schedule and resources for testing.

> EXPLANATION: Certification goals and process control standards
>
> "Certification goals" are targets for final results. "Process control standards" are gauges of intermediate progress toward certification goals. The certification goals answer the question "Is the software currently fit for its intended use?" Process control standards answer the question "Is the software likely to be certified as fit for use on the expected schedule?" In general, certification goals are defined by the customer, process control standards are defined by the developer, and both exist within the context of the predefined certification protocol in the test plan.

Document decisions in the *Increment Certification Report.*

Verification

Verification 1 **Verify that the tests were executed according to the test plan.**

Verification 2 **Verify the correctness of statistical calculations.**

Measurement

Measurement 1

Measure the *Statistical Test Cases* and the results of their execution.

Measure the *Statistical Test* Cases in terms such as the number and size of the test cases, and the execution times for each.

Measure the number and severity of failures reported.

Measurement 2

Measure the Statistical Testing and Certification Process.

Measure the sufficiency of testing in terms such as the coverage of the usage models employed and the statistical similarity between expected usage and tested usage.

Exit

The process is complete when one of the exit criteria is satisfied.

Exit 1

The software increment satisfies certification goals.

Exit 2

The software increment has failed to satisfy certification goals and must be reengineered and reverified before testing can resume.

In either case, the *Statistical Testing Report* and *Increment Certification Report* are completed.

5 Cleanroom Software Engineering Work Products

This section defines the purpose and content of work products produced by the Cleanroom processes.

Cleanroom Engineering Guide

The *Cleanroom Engineering Guide* is created in the Project Planning Process. It defines the adaptation and refinement of the Cleanroom processes to meet project-specific requirements. It includes process definitions, work product definitions, and local policies, procedures, templates, and forms that define how a project will be conducted. It identifies the facilities, hardware and software environments, and tools to support Cleanroom operations, and defines guidelines for their use.
It also defines relationships among Cleanroom processes.

An organization-level *Cleanroom Engineering Guide* constitutes the "standard software process" required in the CMM Level 3 Organization Process Definition KPA. The *Cleanroom Engineering Guide* may be successively refined and elaborated for use by organizational divisions, product lines, and specific projects. At each level, the Guide is tailored for standards, technologies, languages and other aspects of the development environment at that level.

The *Cleanroom Engineering Guide* for a project constitutes the "tailored version of the organization's standard software process" required in the CMM Level 3 Integrated Software Management KPA. The tailored Guide also documents the "plans for the project's software engineering facilities and support tools" required in the CMM Level 2 Software Project Planning KPA.

Configuration Management Plan

See *Software Development Plan.*

Engineering Change Log

The *Engineering Change Log* is created and maintained in the Engineering Change Process. It is the record of all engineering change requests, together with their evaluations, impacts, and status.

Executable System

The *Executable System* is created in the Statistical Testing and Certification process. It is the executable form of the accumulating increments to be used for testing and customer evaluation.

Function Specification

The *Function Specification* is created in the Function Specification Process. It documents 1) software boundaries and interfaces with hardware, other software, and human users and 2) the external view of a system in terms of the mapping of all possible stimuli to their corresponding responses in all possible circumstances of use, including correct and incorrect, frequent and infrequent, and nominal and stress usage conditions.

Based on set theory and function theory, the *Function Specification* is a precise statement of the *Software Requirements* as a mathematical function. The domain of the function is all possible stimulus histories and the range is all correct responses. The mathematical form of the *Function Specification* as a set of mapping rules provides a flexible yet verifiable basis for function decomposition.

From the customer's perspective, the *Function Specification* is the definitive statement of functional requirements for the software. From a development perspective, the *Function Specification* is the top-level black box in the box structure usage hierarchy that is fully realized in the *Increment Design*.

Increment Certification Report

The *Increment Certification Report* is created in the Statistical Testing and Certification Process. It contains values for measures of certification goals (the desired "ends") and measures of process control (the efficiency of "means" based on historical performance). Certification measures may include reliability and confidence, mean time to failure, representativeness of the test case sample, and

other measures of product quality. Process control measures may include reliability growth rate, error rate per unit volume of code, and other measures of process performance.

The *Increment Certification Report* documents the quantitative basis for management decisions about the testing process. Continuation of testing, cessation of testing for engineering change or reengineering, and certification of the software are justified on the basis of product and process measures. The *Increment Certification Report* documents the "analysis of data on defects identified in testing" required in CMM Level 3 Software Product Engineering KPA. It also documents the "results of the project's quantitative process management activities" required in the CMM Level 4 Quantitative Process Management KPA.

Increment Construction Plan

The *Increment Construction Plan* is created in the Increment Planning Process. It specifies the number of increments into which a Cleanroom Project will be divided, the functions that will be implemented in each increment, and the schedule and resources allocated for each increment. The *Increment Construction Plan* is used by management to assign tasks, track progress, and monitor product quality and process control.

The earliest version of the *Increment Construction Plan* may be based on the customer's *Statement of Work* and/or the *Software Requirements*. This version will contain assumptions that will be explored further in the course of preparing the *Risk Analysis Plan* and the *Reuse Analysis Plan*. A sound basis for increment planning will exist when the *Function Specification* and the *Usage Specification* have been prepared; the *Increment Construction Plan* should be considered preliminary until these two work products are available. The *Increment Construction Plan* is also influenced by the *Software Architecture*. The *Increment Construction Plan* documents the "software life cycle with predefined stages of manageable size" required in the CMM Level 2 Project Planning KPA.

Increment Design

The *Increment Design* is created in the Increment Design Process. It is the box structure implementation of a set of functions named in the *Increment Construction Plan* and defined in the *Function Specification*. The *Increment Design* is a hierarchy of components in which each component is represented in black box (history-based), state box (state-based), and clear box (procedure-based) forms.

Clear boxes in the *Increment Design* may contain new black boxes which are either implemented or stubbed. In each *Increment Design*, some previously stubbed functions are implemented.

Increments are cumulative. An *Increment Design* is the sum of all specification, design, and code to date. The final *Increment Design* is the completed product.

Increment Evaluation Report

The *Increment Evaluation Report* is originated by the customer. It is the customer's documentation of feedback from increment execution and evaluation.

Increment Test Plan

The *Increment Test Plan* is created in the Usage Modeling and Test Planning Process. It contains all information needed by the certification team for the Statistical Testing and Certification Process, including schedules, staffing, training, hardware and software environments, data collection forms, test case evaluation procedures, certification goals, and statistical models. The *Increment Test Plan* is the "plan for system testing to demonstrate that the software satisfies its requirements" required in the CMM Level 3 Software Product Engineering KPA.

Increment Verification Report

The *Increment Verification Report* is created in the Correctness Verification Process. It is the record of experience during the Correctness Verification Process, including staff members participating, number of verification sessions, time spent in each session, faults found during each session, and any other information relevant to assessment of the correctness of the design. Data for sessions in which engineering changes are verified are also included in the *Increment Verification Report*.

In addition to the raw data above, the *Increment Verification Report* may contain other measures that provide indications of process control. Such measures may include percentage of engineering changes that are found to be incorrect, the distribution of faults with regard to severity and type, and number of faults found per unit volume of code.

The *Increment Verification Report* constitutes the "data on the conduct and results of peer reviews" required in CMM Level 3 Peer Reviews KPA. It also documents the "data on defects identified in peer reviews" required in the CMM Level 3 Software Product Engineering KPA.

Measurement Plan

See *Software Development Plan.*

Performance Improvement Plan

The *Performance Improvement Plan* is created in the Performance Improvement Process. It documents plans to improve team performance by refining the current *Cleanroom Engineering Guide* and/or exploring the use of new software technologies.

The *Performance Improvement Plan* contains an analysis of the cause of each failure that occurred during statistical testing, and plans to prevent the recurrence of the underlying problem. It also documents the comparison of current performance with planned or historical performance for the measures defined in the *Measurement Plan.*

The *Performance Improvement Plan* documents the "causal analysis meetings" and "revisions to the project's defined software process resulting from defect prevention actions" required in the CMM Level 5 Defect Prevention KPA; the "incorporation of appropriate new technologies into a project's defined software process" required in the CMM Level 5 Technology Change Management KPA; and the "plan for software process improvement" required in the CMM Level 5 Process Change Management KPA.

Project Mission Plan

See *Software Development Plan.*

Project Organization Plan

See *Software Development Plan.*

Project Record

The *Project Record* is created in the Project Management Process and updated in all processes. It documents actions, reviews, decisions, measures and other events throughout a project.

The *Project Record* contains formal documents, such as contracts and reports, and informal correspondence, such as meeting notes or records of phone conversations. It is the archive of documentation about all project events that are not captured in other Cleanroom work products. It is a flexible, tailorable work product that is the Cleanroom vehicle for fulfilling project documentation requirements not met by other work products.

Reengineering Plan

The *Reengineering Plan* is created in the *Software Reengineering Process*. The *Reengineering Plan* documents the tasks, schedules, and resources required to prepare existing artifacts for reuse in the current project.

The *Reengineering Plan* elaborates the technical aspects of the *Reuse Analysis Plan*, i.e., defines specific investigations required to make decisions about the reusability of a component and/or adaptations required to reuse a component in the current system.

Reuse Analysis Plan

See *Software Development Plan*.

Reengineered Software

The *Reengineered Software* is created in the Software Reengineering Process. It consists of specification, design, code, usage models and/or testing artifacts produced in the reengineering of reused components.

Risk Analysis Plan

See *Software Development Plan*.

Schedule and Resource Plan

See *Software Development Plan*.

Software Architecture

The *Software Architecture* is created in the Architecture Specification Process. *The Software Architecture* identifies 1) the conceptual architecture expressed in terms of principal software components and their relationships, 2) the module architecture expressed in terms of layers of functional decomposition, and 3) the execution architecture expressed in terms of dynamic software operation [Soni 95].

The *Software Architecture* serves as a vehicle for analyzing application and service domains, reference architectures, reusable assets, communication protocols, standards, and software design strategies. The *Software Architecture* is a principal input to the Increment Planning and Increment Design Processes.

Software Development Plan

The *Software Development Plan* is created in the Project Planning Process. It is used in the Project Management Process for task initiation, performance tracking, and quantitative process management. The *Software Development Plan* is the "software project plan" required in CMM Level 2 Software Project Planning KPA and the "software development plan" to be used in CMM Level 2 Software Project Tracking and Oversight KPA. The *Software Development Plan* consists of the following project management plans.

The ***Project Mission Plan*** defines the overall mission, goals, and objectives of the system and the Cleanroom development project.

The ***Project Organization Plan*** defines the structure, responsibilities, and relationships of the Cleanroom project organization and peer organizations. The *Project Organization Plan* is the "documented plan to communicate intergroup commitments and coordinate and track the work performed" required by CMM Level 3 Intergroup Coordination KPA.

The ***Work Product Plan*** defines the Cleanroom work products to be produced by the project. The *Work Product Plan* constitutes the "identification of software work products" required in CMM Level 2 Software Project Planning KPA.

The ***Schedule and Resource Plan*** defines estimates for overall tasks, schedules, milestones, budgets, and resources for Cleanroom work product development. The *Schedule and Resource Plan* documents the "estimates of size, effort,

schedule, cost, and critical computer resources" required in CMM Level 2 Software Project Planning KPA.

The ***Measurement Plan*** defines product and process measurements, standards, and goals for managing the project, including those for Cleanroom software certification and statistical process control. The *Measurement Plan* defines the "plan for quantitative process management" and the "strategy for data collection and analysis" required in CMM Level 4 Quantitative Process Management KPA.

The ***Reuse Analysis Plan*** identifies sources of reusable assets and asset acquisition and evaluation tasks. It also identifies opportunities to reuse domain models, reference architectures, software specifications, designs, code, and usage models. The *Reuse Analysis Plan* is a management plan for identification of assets. A related work product, the *Reengineering Plan*, is a technical plan for evaluation and adaptation of assets.

The ***Risk Analysis Plan*** defines methods for risk analysis, identifies project risks, and describes strategies for risk management and avoidance. The *Risk Analysis Plan* constitutes the "identification, assessment, and documentation of risks associated with the cost, resource, schedule, and technical aspects of the project" required in the CMM Level 2 Software Project Planning KPA.

The ***Standards Plan*** identifies and defines the application of external standards that will be used in the project.

The ***Training Plan*** identifies project training requirements, including training in the application domain, development environments, and Cleanroom technology and processes. This plan is the "training plan" required in CMM Level 3 Training Program KPA.

The ***Configuration Management Plan*** defines requirements for change control of designated work products. This plan is the "software configuration management plan" required in the CMM Level 2 Software Configuration Management KPA.

Software Requirements

The *Software Requirements* document is created in the Requirements Analysis Process. It defines the functional, usage, performance, and environment requirements for a software system to be developed under the Cleanroom process. Included among requirements are operational constraints such as dependencies on other systems, capacity requirements, and reliability requirements. *The Software Requirements* are typically documented in user

terms. It is the principal input to the Function Specification and Usage Specification Processes, where requirements are defined in the more precise terms essential to software development and certification.

The *Software Requirements* are the "documentation of allocated requirements" required in the CMM Level 2 Requirements Management KPA.

Standards Plan

See *Software Development Plan.*

Statement of Work

The *Statement of Work* is originated by the customer. It is the "documented and approved statement of work for the software project" required in the CMM Level 2 Software Project Planning KPA.

Statistical Test Cases

The *Statistical Test Cases* are created in the Usage Modeling and Test Planning Process. *Statistical Test Cases* are randomly generated from a usage model for use in statistical testing of an increment. Once generated, test cases may undergo post-processing to add information for human testers or an automated test tool. Such information may include additional instructions (e.g., events to initiate in the background), invocation of independent data feeds, or pointers to the relevant "oracle" for evaluation of responses.

Each statistical test case is a complete usage scenario given as a sequence of user inputs, beginning with a predefined initial event and ending with a predefined terminal event. The *Statistical Test Cases* become a "script" for testing, and may be annotated during testing to record responses and their evaluations.

Statistical Testing Report

The *Statistical Testing Report* is created in the Statistical Testing and Certification Process. It is the record of experience in testing, and includes staff members participating, number of compilation sessions, faults found in compilation, number of testing sessions, number of test cases run in each session, failures observed in test cases, faults found during investigation of failures, time required to correct each fault, and any other information relevant to assessment of the correctness of the executing software.

The *Statistical Testing Report* documents the "data on defects identified in testing" and the "performance of system testing to demonstrate that the software satisfies its requirements" required in CMM Level 3 Software Product Engineering KPA.

Training Plan

See *Software Development Plan*.

Usage Models

The *Usage Models* are created in the Usage Modeling and Test Planning Process. A usage model is a formal representation of software use, often expressed as a Markov chain. It defines the usage states of the software and the probabilities of transitions between usage states. When software is to be certified for normal operational use, usage probabilities are based on expected use; when the customer requires certification for other usage conditions, the probabilities reflect those conditions.

Usage model analysis provides numerous insights into software usage characteristics that are useful in making management and technical decisions. Usage models are also used as test case generators.

Usage Specification

The *Usage Specification* is created in the Usage Specification Process. It is a description of the expected users, usage scenarios, and usage environments of the software. It contains definitions of high-level usage models that record this information, as well as the results of model analysis for management decision making.

Work Product Plan

See *Software Development Plan*.

6 References

[Basili 94] Basili, V.R. & Green, S.E. "Software Process Evolution in the SEL." *IEEE Software 11*, 7 (July 1994): 58-66.

[Dyer 90] Dyer, M. & Kouchakdjian, A. Ch. 9, "Correctness Verification: Alternative to Structural Testing." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Ett 96] Ett, W.H. & Trammell, C.J. "A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering." Available WWW URL: <http://source.asset.com/stars/loral/cleanroom/guide.html> (1996).

[Hausler 92] Hausler, P.A. "A Recent Cleanroom Success Story: The Redwing Project." *Proceedings of the Seventeenth Annual Software Engineering Workshop.* Greenbelt, Md., December 1992. Greenbelt, Md.: NASA, Goddard Space Flight Center., 1992.

[Hausler 94] Hausler, P.A.; Linger, R.C.; & Trammell, C.J. Ch. 1, "Adopting Cleanroom Software Engineering with a Phased Approach." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Head 94] Head, G.E. Ch. 11, "Six Sigma Software Using Cleanroom Software Engineering Techniques." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Linger 79] Linger, R.C; Mills, H.D.; & Witt, B.I. *Structured Programming: Theory and Practice.* Reading, Ma.: Addison-Wesley, 1979.

[Linger 88] Linger, R.C. & Mills, H.D. "A Case Study in Cleanroom Software Engineering: The IBM COBOL Structuring Facility." *Proceedings of 12th Annual International Computer*

Software and Applications Conference. 1988. Los Alamitos, Ca.: IEEE Computer Society Press, 1988.

[Linger 93] Linger, R.C. "Cleanroom Software Engineering for Zero-Defect Software." *Proceedings of 15th International Conference on Software Engineering.* Baltimore, Md., May 17-21, 1993. Los Alamitos, Ca.: IEEE Computer Society Press, 1993.

[Linger 94] Linger, R.C. "Cleanroom Process Model." Ch. 6, *Cleanroom Software Engineering: A Reader,* Oxford, England: Blackwell Publishers, 1996.

[Mills 86] Mills, H.D.; Linger R.C.; & Hevner, A.R. *Principles of Information Systems Analysis and Design.* New York: Academic Press, 1986.

[Mills 87] Mills, H.D.; Linger, R.C.; & Hevner, A.R. Ch. 7, "Box-Structured Information Systems." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Mills 88] Mills, H.D. Ch. 8, "Stepwise Refinement and Verification in Box-Structured Systems." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Mills 92] Mills, H.D. "Certifying the Correctness of Software." *Proceedings of 25th Hawaii International Conference on System Sciences.* Kauai, Hawaii, January 7-10, 1992. Los Alamitos, Ca.: IEEE Computer Society Press, 1992.

[Poore 93] Poore, J.H.; Mills, H.D.; & Mutchler, D. Ch. 5, "Planning and Certifying Software System Reliability." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Poore 95a] Poore, J.H. "Usage Testing as Engineering Practice." *Proceedings of the 2nd European Industrial Symposium on Cleanroom Software Engineering.* Berlin, Germany, March 1995. Lund, Sweden: Q-Labs, 1995.

[Poore 95b] Poore, J.H.; Walton, G.H.; & Whittaker, J.A. "A Mathematical Programming Approach to the Representation and Optimization of Markov Usage Models." Technical Report, Department of Computer Science, University of Tennessee, Knoxville , Tn., 1995.

[Poore 96a] Poore, J.H. Ch. 4, "The Cleanroom Approach to Six Sigma: Combining Information." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Poore 96b] Poore, J.H. & Trammell, C.J. *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Sherer 96] Sherer, S.W.; Kouchakdjian, A.; & Arnold, P.G. "Experience Using Cleanroom Software Engineering." *IEEE Software 13*, 3 (May 1996): 69-76.

[Soni 95] Soni, D.; Nord, R.; & Hofmeister, C. "Software Architecture in Industrial Applications." *Proceedings, 17th International Conference on Software Engineering.* Seattle, Wa., April 23-30, 1995. New York: Association for Computing Machinery, 1995.

[Trammell 94] Trammell, C.J. & Poore J.H. "Experimental Control in Software Reliability Certification." *Proceedings of the Nineteenth Annual Software Engineering Workshop.* College Park, Md., October 31-November 1, 1994. College Park, Md.: NASA/GSFC Software Engineering Laboratory, 1994.

[Trammell 95] Trammell, C.J. "Quantifying the Reliability of Software: Statistical Testing Based on a Usage Model." *Proceedings of the Second IEEE International Symposium on Software Engineering Standards.* Montreal, Quebec, Canada, August 21-25, 1995. Los Alamitos, Ca.: IEEE Computer Society Press, 1995.

[Trammell 96] Trammell, C.J.; Pleszkoch, M.G.; Linger, R.C.; & Hevner, A.R. "The Incremental Development Process in Cleanroom Software Engineering." *Decision Support Systems 17*, 1 (April 22, 1996) 55-71.

[Walton 95a] Walton, G.H.; Poore, J.H.; & Trammell, C.J. Ch. 15, "Statistical Testing Based on a Software Usage Model." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Walton 95b] Walton, G.H. & Poore J.H. "Measuring Complexity and Coverage of Software Specifications." Technical Report, Department of Computer Science, University of Tennessee, Knoxville, Tn., 1995.

[Whittaker 93] Whittaker, J.A. & Poore, J.H. Ch. 13, "Markov Analysis of Software Specifications." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Whittaker 94a] Whittaker, J.A. & Thomason, M.G. Ch. 14, "A Markov Chain Model for Statistical Software Testing." *Cleanroom Software Engineering: A Reader.* Oxford, England: Blackwell Publishers, 1996.

[Whittaker 94b] Whittaker, J.A. & Agrawal, K.K. "A Case Study in Software Reliability Measurement." *Proceedings of the Seventh International Quality Week*, San Francisco, Ca., May 17-20, 1994. San Francisco, Ca.: Software Research, Inc., 1994.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-96-TR-022	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-96-022
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (city, state, and zip code) HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116
8a. NAME OF FUNDING/ SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESC/AXS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-95-C-0003
8c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO: 63756E PROJECT NO.: N/A TASK NO: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) Cleanroom Software Engineering Reference Model	
12. PERSONAL AUTHOR(S) Richard C. Linger, Carmen J. Trammell	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) November 1996 15. PAGE COUNT 130 pp.
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	box structures, cleanroom software engineering, correctness verification, reference models, reliability certification, software architecture, software reeengineering, software specification, statistical quality control, statistical testing, usage models

19. ABSTRACT (continue on reverse if necessary and identify by block number)

Cleanroom software engineering is a theory-based, team-oriented process for development and certification of high-reliability software systems under statistical quality control. A principal objective of the Cleanroom process is development of software that exhibits zero failures in use. The Cleanroom name is borrowed from hardware Cleanrooms, with their emphasis on rigorous engineering discipline and focus on defect prevention rather than defect removal. Cleanroom combines mathematically-based methods of software specification, design, and correctness verification with statistical, usage-based testing to certify software fitness for use. Cleanroom projects have reported substantial gains in quality and productivity. This report defines the Cleanroom Software Engineering Reference Model.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ❑ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution	
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (incl. area code) (412) 268-7631	22c. OFFICE SYMBOL ESC/AXS (SEI)